VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Dokazování pomocí přirozené dedukce v jazyku TIL-Script
Natural Deduction System for the TIL-Script Language

2019 Autor: Vojtěch Patschka

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Vojtěch Patschka**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Dokazování pomocí přirozené dedukce v jazyku TIL-Script
Natural Deduction System for the TIL-Script Language

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je shrnout teorii Transparentní intensionální logiky (TIL) a její komputační varianty TIL-Script. Student naprogramuje interaktivní nástroj pro dokazování pomocí přirozené dedukce v TIL-Script.

Cíle práce:
1. Text práce bude obsahovat shrnutí teorie TIL a syntax jazyka TIL-Script.
2. Práce bude obsahovat seznam pravidel pro přirozenou dedukci.
4. Součástí práce bude analýza, návrh a implementace nástroje pro dokazování platnosti úsudků formalizovaných v jazyku TIL-Script.

Seznam doporučené odborné literatury:

[1] Duží M., Materna P. (2012): TIL jako procedurální logika (průvodce zvídavého čtenáře Transparentní intensionální logikou). Aleph Bratislava 2012, ISBN 978-80-89491-08-7
[2] Duží M., Jespersen B. and Materna P. (2010): Procedural Semantics for Hyperintensional Logic. Foundations and Applications of Trasnsparent Intensional Logic. First edition. Berlin: Springer, series Logic, Epistemology, and the Unity of Science, vol. 17, ISBN 978-90-481-8811-6.
[3] Švejdar, V.: Logika - neúplnost, složitost a nutnost, Academia, 2002, ISBN: 80-200-1005-X

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Mgr. Marek Menšík, Ph.D.**

Datum zadání: 01.09.2018

Datum odevzdání: 30.04.2019

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou/diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 29.4.2019

Bc. Vojtěch Patschka

Abstrakt

V mé práci se zabývám vytvořením interaktivního nástroje pro dokazování pomocí přirozené dedukce v jazyce TIL-Script. Pro použití nástroje shrnu základy logických kalkulů, teorie Transparentní intenzionální logiky a její komputační variantu TIL-Script. V práci uvádím úplný seznam všech pravidel pro přirozenou dedukci, které je možno v nástroji použít a výslednou analýzu, návrh a implementaci nástroje.

Klíčová slova

přirozená dedukce, logika, TIL, axiom, odvozovací pravidlo, důkaz

Abstract

My work is to create an interactive tool for proving using natural deduction in TIL-Script language, To use the tool I will summarize the basics of logical calculus, the theory of Transparent intensional language and it's computational variant TIL-Script. In the work I present complete list of all the rules for natural deduction that can be used in the tool and final analysis, design and implementation of the tool.

Keywords

natural deduction, logic, TIL, axiom, deduction rule, proof

Obsah

Seznam použitých symbolů a zkratek

TIL – transparentní intenzionální logika

PL1 – predikátová logika prvního řádu

Seznam ilustrací

Seznam tabulek

1. Úvod

V mé práci jsem se zabýval vytvořením nástroje pro přirozenou dedukci v jazyce TIL-Script. Jazyk TIL-Script je komputační variantou Transparentní intenzionální logiky (dále TIL), což je parciální typovaný hyperintenzionální lambda kalkul, pro analýzy vět přirozeného jazyka.

V první části mé práce představím co je to formální systém a jak souvisí s přirozenou dedukcí, jaké jsou techniky přirozené dedukce a jak vypadá přirozená dedukce v logice výrokové a v predikátové logice prvního řádu. Představím pravidla a nedostatečnosti těchto logik pro analýzu.

Následně se zabývám popisem TILu, od jeho charakteristik po jednotlivé koncepty a základní pojmy, které TIL tvoří. Uvedu, jaké TIL používá objektové a intenzionální báze, jaké má typy, kontexty a co jsou konstrukce. Poté představím samotný TIL-Script a jeho rozdíly oproti TIL.

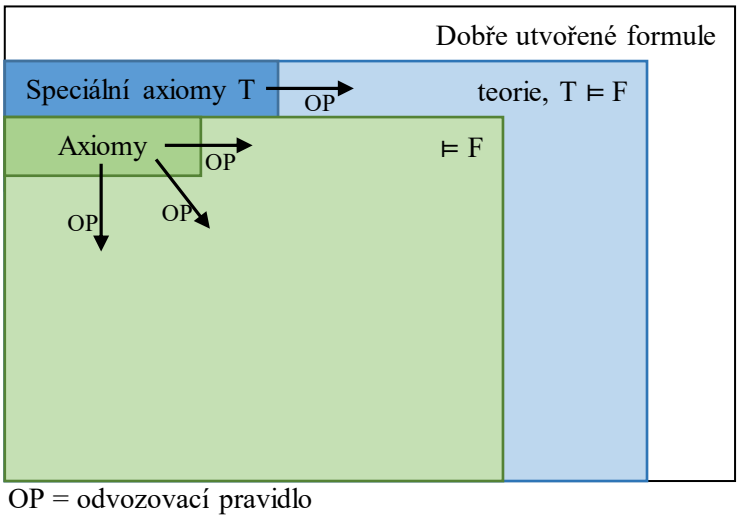
Po představení jazyka TIL-Script ukážu pravidla přirozené dedukce, které na výrazy v TIL-Scriptu můžeme aplikovat, spolu s příklady. Kapitolu zakončím příkladem důkazu, který je program schopen zpracovat.

Poslední částí je analýza a uživatelské rozhraní výsledného nástroje.

2. Logický kalkul

Tato kapitola vychází ze zdroje 3.

Logický kalkul slouží k získání a ověřování logické pravdivosti formulí. Rozdělujeme axiomatické a předpokladové logické kalkuly. Logický kalkul se skládá z jazyka, axiomů a odvozovacích pravidel. Pomocí jazyka vytváříme formule, což je konečná posloupnost vytvořená ze symbolů jazyka. Axiomy jsou tautologie, které musí být pravdivé a v rámci kalkulu je nedokazujeme. Odvozovací pravidla jsou funkce, které můžeme provádět na formulích jazyka. Pravidla volíme takové, aby zachovávaly pravdivost, případně aby celý důkazový postup zachovával pravdivost, a tudíž abychom mohli ověřovat tautologičnost. Jazyk, axiomy a všechny operace, které na jazyku provádíme jsou čistě syntaktické a jazyk je neinterpretovaný. Proto můžeme říct, že logický kalkul má syntaktickou bázi. Právě interpretace, která není součástí operací potřebujeme, pokud chceme přiřadit formulím jejich význam.



Obr. 1 - Ukázka vztahu mezi množinami tvořící logický kalkul

2.1. Formální jazyk

Oproti přirozenému jazyku formální jazyky nevznikly přirozeně a živelně v průběhu lidského vývoje, ale jsou striktně definovány umělou dohodou, díky které můžeme přesně určit, co výrazy označují, a tedy jakou mají interpretaci. Tato vlastnost explicitně definované interpretace je jedním z hlavních důvodů používání formálních jazyků v logice a ve výpočetních modelech. Formální jazyky tak umožňují abstrahovat od určitých rysů přirozeného jazyka, které mohou zanést nepřesnosti a chyby do modelu vlivem nesprávného pochopení. Jazyk logiky je tvořen množinou dobře utvořených formulí.

2.2. Axiomy

Axiomy jsou výchozí formule, které se nedokazují a jsou základním stavebním prvkem celého formálního systému. Axiomy můžeme považovat za odvozovací pravidla s nulovým počtem předpokladů, tedy jsou vždy pravdivé. Množina axiomů tvořící systém nesmí být nikdy prázdná a musí existovat postup v rámci množiny dobře utvořených formulí, který jednoznačně odpoví, zda se jedná o axiom nebo ne. Pokud by množina axiomů tato pravidla nesplňovala, nemohli bychom v takovém formálním systému nic dokazovat. Pokud je množina axiomů konečná, pak rozhodnutí, jestli je dobře utvořená formule axiom triviální. Není neobvyklé, že je množina axiomů definována rekurzivně, a tudíž bude tato množina nekonečná. Tuto rekurzivní definici můžeme zapsat pomocí tzv. axiomových schémat. Alternativní způsob definice nekonečného počtu axiomů je použít popis algoritmu pro jejich vytváření. Zvolené axiomy jsou tautologie neboli jsou pravdivé v každé interpretaci. Axiomy bývají

voleny tak, aby byly navzájem nezávislé, tedy aby se axiomy nedaly vyjádřit z jiných, protože by jinak byly nadbytečné.

2.3. Odvozovací pravidla

Odvozovací nebo též dedukční pravidla definují, jak upravovat dobře utvořené formule formálního systému na jiné dobře utvořené formule. Obvykle pravidla zachovávají pravdivost, což znamená, že pokud na tautologii aplikujeme takovéto pravidlo, výsledný výraz bude opět tautologií. Pokud v průběhu důkazu použijeme pravidlo, které nezachovává pravdivost, musíme zajistit korektnost celého postupu. Obvykle je množina odvozovacích pravidel tvořena několika pravidly, ale můžeme se i setkat se systémy s pouze jedním pravidlem. Odvozovací pravidla, která používáme, mají obecně tento tvar:

$$P_1, P_2, \dots, P_n \vdash F_1, F_2, \dots, F_m.$$

Z tohoto pravidla vidíme, že pokud platí všechny předpoklady $P_1 \dots P_n$, pak platí každá z formulí $F_1 \dots F_m$, a tudíž pokud dokážeme všechny předpoklady, které jsou na levé straně pravidla, jakákoli formule na pravé straně pravidla může být také brána za dokázanou.

Pomocí posloupností odvozovacích pravidel aplikovaných na axiomy nebo z nich vycházející dobře utvořené formule můžeme vytvářet důkazy. Výsledkem důkazu a jeho posledním krokem je formule, která je nazývána teorém.

2.4. Vlastnosti kalkulu

U formálních systémů rozlišujeme několik důležitých vlastností, které jsou potřeba nebo žádané, abychom byli schopni pomocí formálního systému dokazovat, případně i dokazování automatizovat.

Pro různé jazyky lze vytvořit formální systémy, ale ne vždy lze najít takový systém, který by splňoval všechny vlastnosti. Pro výrokovou logiku je možné vytvořit takový kalkul, který je konzistentní, korektní, úplný. Kalkuly pro predikátovou logiku mohou být pouze parciálně rozhodnutelné v problému logické pravdivosti. Kvůli parciální rozhodnutelnosti logické pravdivosti PL1 můžeme v konečném počtu kroků rozhodnout, zda je formule tautologií, ale pokud není, můžeme to zjišťovat do nekonečna. Pro predikátovou logiku druhého řádu již nenajdeme konzistentní systém, který by byl úplný nebo rozhodnutelný (ani parciálně).

2.4.1. Korektnost

Korektnost kalkulu je definována tak, že každá v systému dokazatelná formule, musí být logicky pravdivá. Formálně: Platí-li $\vdash A$, pak musí platit $\vDash A$. Korektnost můžeme dokázat tak, že dokážeme, že všechna odvozující pravidla zachovávají pravdivost.

2.4.2. Úplnost

Kalkul je úplný, pokud v něm můžeme dokázat všechny logicky pravdivé formule. Formálně: Platí-li $\vDash A$, pak musí platit $\vdash A$.

2.4.3. Konzistence

Jedním z důležitých požadavků na formální systém je konzistence, pro kterou musíme ukázat, že existuje formule, která není v systému dokazatelná. Pokud ukážeme, že existuje formule, kterou nemůžeme dokázat, znamená to, že systém není sporný, jelikož ve sporném systému dokážeme vše[1]. Typickými formulemi, u kterých chceme ukázat, že nejdou dokázat je konjunkce formule a její negace ($A \wedge \neg A$), případně negace implikace stejné formule ($\neg(A \supset A)$).

[1] Tento fakt se dá nejlépe ukázat tím, že pokud máme jakoukoli formuli, tak můžeme dělat důkaz sporem, ale tím, že v systému již spor je tak formule musí být pravdivá.

2.4.4. Rozhodnutelnost problému logické pravdivosti

Problém logické pravdivosti pro kalkul je rozhodnutelný, pokud existuje algoritmus, jehož vstupem je dobře utvořená formule a výstupem je informace, jestli daná formule je dokazatelná v systému nebo ne v konečném počtu kroků. Problém může být v kalkulu parciálně rozhodnutelný, pokud je možné zjistit, zda je formule dokazatelná v konečném počtu kroků, ale pokud není dokazatelná nemusí se nikdy zastavit.

2.5 Teorie

Pokud chceme dokazovat pravdivost výroků v určité teorii, stanovujeme navíc množinu speciálních axiomů, které ji charakterizují. Takovéto speciální axiomy nemusí být pravdivé ve všech interpretacích, ale musí být pravdivé v interpretaci teorie, kterou dokazujeme.

2.5.1 Úplnost teorie

Teorie je úplná, pokud můžeme dokázat každou dobře utvořenou formuli nebo její negaci.

3. Přirozená dedukce

Tato část vychází ze zdroje 3.

Jedna z možností, kterou můžeme vytvořit důkazový kalkul logiky, je použití přirozené dedukce. Jedná se o předpokladový formální systém.

3.1. Přirozená dedukce ve výrokové logice

V přirozené dedukci ve výrokové logice jsou definována následující výchozí dedukční pravidla, která se jako taková nedokazují:

Axiomy: $\vdash A \vee \neg A, \vdash A \supset A$
Zavedení konjunkce: $A, B \vdash A \wedge B$
Eliminace konjunkce: $A \wedge B \vdash A, B$
Zavedení disjunkce: $A \vdash A \vee B$ nebo $B \vdash A \vee B$
Eliminace disjunkce: $A \vee B, \neg A \vdash B,\ A \vee B, \neg B \vdash A$
Zavedení implikace: $B \vdash A \supset B$
Eliminace implikace (také latinsky „modus ponens“): $A \supset B, A \vdash B$
Zavedení ekvivalence: $A \supset B, B \supset A \vdash A \equiv B$
Eliminace ekvivalence: $A \equiv B \vdash A \supset B, B \supset A$

Příklad: Pokud je středa a je pěkně, jsem ve škole a studuji. Je středa a je pěkně. |– Jsem ve škole a studuji.

s – je středa
p – je pěkně
v – jsem ve škole
t – studuji

1.	(s ∧ p) ⊃ (v ∧ t)	Předpoklad 1
2.	s ∧ p	Předpoklad 2
3.	v ∧ t	Eliminace implikace na 1 a 2

3.2. Přirozená dedukce v predikátové logice 1. řádu

Jelikož predikátová logika využívá více expresivní jazyk než logika výroková, je i metoda přirozené dedukce pro tuto logiku zobecněním metody přirozené dedukce pro výrokovou logiku. V souvislosti s tím rozšíříme výchozí dedukční pravidla výrokové logiky o další pravidla, která pracují s kvantifikátory. Tato nová pravidla jsou zavedení a eliminace obecného a existenčního kvantifikátoru:

Zavedení obecného kvantifikátoru: $A(x) \vdash \forall x A(x)$

Toto pravidlo smíme použít pouze tehdy, kdy A(x) není odvozeno z předpokladu, který obsahuje *x* jako volnou proměnnou.

Eliminace obecného kvantifikátoru: $\forall x A(x) \vdash A(x/t)$

Pravidlo lze použít pouze v případě, kdy je term *t* substituovatelný za proměnnou *x* v dané formuli A(*x*). Výsledkem korektní substituce je formule A(*x/t*).

Zavedení existenčního kvantifikátoru: $A(x/t) \vdash \exists x A(x)$

Eliminace existenčního kvantifikátoru: $\exists x A(x) \vdash A(x/c)$

Toto pravidlo nezachovává pravdivost, a pokud ho budeme chtít použít v přímém důkazu, musíme poté dále v důkazu opět existenční kvantifikátor zavést, aby byl korektní celý důkazový postup. Při použití tohoto pravidla musíme vždy za proměnnou x substituovat konstantu c pro různé formule A. Ovšem jsou-li ve formuli A také volné proměnné $y_1,\ldots,y_n$, které jsou v dosahu všeobecných kvantifikátorů je nutno pravidlo zobecnit na tento tvar:

$$\exists x A(x, y_1, \ldots, y_n) \vdash A(x/f(y_1, \ldots, y_n), y_1, \ldots, y_n)$$

Ze zobecněného pravidla vidíme, že za proměnnou x již nemůžeme substituovat konstantu, ale celý funkční term f s argumenty $y_1,\ldots,y_n$. Pokud toto pravidlo použijeme vícekrát pro různé formule A, musíme pro substituci vždy použít jinou funkci $f(y_1,\ldots,y_n)$.

Příklad: Jana chce být královnou Anglie. Královna Anglie je Alžběta. Královna Anglie existuje. |– Jana chce být Alžbětou.
j – Jana, a – Alžběta, n – Anglie, k(x) – královna x, C(a, b) – a chce být b

1. C(j, k(n)) Předpoklad 1
2. k(n) = a Předpoklad 2
3. C(j, a) Nahrazení identických výrazů

U posledního příkladu vidíme, že jsme dokázali něco, co očividně není pravda. Je zřejmé, že osoba, která chce být královnou se nechce převtělit do osoby, která je královna. Problémem je, že první věta mluví o touze zastávat určitý úřad, ale druhá věta o osobě okupující tento úřad. Bohužel v predikátové logice 1. řádu neumíme tento případ ošetřit, a proto se musíme obrátit na silnější systémy jako je TIL.

3.3. Techniky přirozené dedukce

3.3.1. Přímý důkaz

Přímý důkaz je posloupnost výrazů, které zahrnují předpoklady, axiomy a výrazy vzniklé aplikací odvozovacích pravidel na předcházející členy posloupnosti. Pro zjednodušení našeho důkazu můžeme použít teorémy, tedy takové formule, k nimž existuje důkaz bez předpokladů. Pokud chceme tyto teorémy použít v našem důkazu, musíme pravdivost teorému dokázat.

3.3.2. Důkaz sporem

Při důkazu sporem, pokud dokazujeme úsudek přidáváme k množině předpokladů i negaci závěru, při důkazu tautologie ji znegujeme. V průběhu důkazu se budeme snažit ukázat, že dojde ke sporu a tím pádem, že množina předpokladů s negovaným závěrem je sporná. Díky tomu víme, že konjunkce předpokladů s negovaným závěrem je kontradikce, a tudíž původní závěr vychází z předpokladů. Typickým sporem bývá nalezení výrazu a jeho negace v průběhu tvoření důkazu.

3.3.3. Technika hypotetických důkazů

Hypotetický předpoklad můžeme uvést za počátečními výrazy, které tvoří důkaz. Pokud se nám podaří dokázat, že z tohoto hypotetického předpokladu lze odvodit jiný výraz, pak můžeme k důkazu připojit řádný předpoklad, kde hypotetický předpoklad implikuje odvozený výraz. Pokud je ovšem odvozený výraz ve sporu s některým z předpokladů, můžeme za řádný předpoklad uvést pouze negaci předpokladu hypotetického.

3.3.4. Technika větveného důkazu z hypotéz

Pokud máme v našem důkazu výraz, jež je disjunkcí jednotlivých podvýrazů, lze takový výraz dokázat dokázáním jednotlivých podvýrazů na základě dodatečných hypotetických předpokladů každého podvýrazu.

4. Transparentní intenzionální logika

Tato kapitola vychází ze zdroje 1.

Transparentní intenzionální logika (dále TIL) je formální jazyk, který byl vytvořen českým logikem Pavlem Tichým. V současné době jde o jeden z nejkomplexnějších a nejvíce expresivních logických systémů a jako takový nám umožňuje přesnější analýzu a zpracování přirozeného jazyka, než některé jiné, méně expresivní, logické systémy.

TIL je parciální a hyper-intenzionální modifikací typovaného lambda kalkulu. Základem jazyka jsou tedy procedury vyjadřující funkce, které mohou být parciální. Parciální funkce nemusí být definována pro všechny možné vstupy a v takovém případě nic nevrací.

4.1. Charakteristiky TIL

4.1.1. Princip kompozicionality

Jde o princip, jež říká, že význam výrazu je určen významy jeho složek. Pokud tedy chceme analyzovat větu z přirozeného jazyka, tak součástí výrazu, jež mapuje její význam, musí být všechny podvýrazy, které se v dané větě vyskytují, a naopak nesmí obsahovat podvýrazy neobjevující se v analyzované větě.

4.1.2. Antikontextualismus

Antikontextualismus je odpovědí na určité nevýhody kontextualismu, například absurdní důsledky kontextuálních logik bez uvedení určitého kontextu. Problém nastává, když takových kontextů je neúnosně velké množství. TIL je antikontextuální, jelikož význam daného výrazu je vždy stejný ve všech kontextech.

4.1.3. Platonismus, realismus

Názor, že vedle a nad hmotnými objekty jsou pojmy, myšlenky a funkce se nazývá platonismem.

Idea realismu považuje myšlenky a jejich pravdivost za zcela samostatné a nezávislé na konkrétním vyjádření i jazyce.

V TIL jsou tyto myšlenky reflektovány způsobem analýzy.

4.1.4. Antiformalismus

Formalismus samotný je v TIL viděn spíše jako prostředek pro zjednodušení vyjádření abstraktních mimojazykových objektů.

4.1.5. Antiaktualismus

Jako mnoho intenzionálních logik i tady pracujeme s množinou možných světů, kde svět je časovou posloupností množin empirických faktů, které v něm platí. Neplatí tedy, že by sémantická pravdivostní hodnota empirického výrazu byla určena pouze jediným aktuálním (neboli skutečným) světem. Pro tento přístup existují především dva velmi závažné problémy, které se objevují v logikách, které jsou aktualistické.

Prvním z těchto argumentů je problém tzv. empirické vševědoucnosti. Problém se objeví, pokud za význam výrazu stanovíme určité individuum, což by znamenalo, že víme, který svět je aktuální, a tudíž bychom nemuseli empiricky zkoumat stav světa, byli bychom vševědoucí.

Dalším problémem je, že kvůli pravdivosti ve všech světech a časech ztrácíme informaci, kterou chceme logikou přenést. Tento problém se nazývá jalovost.

4.2. Základní pojmy TIL

Kvůli potřebě nahradit intuitivně chápané výrazy přesně definovanými pojmy existuje v TIL objektová a intenzionální báze. Objektová báze je soubor množin, který určuje způsob tvorby funkcí, ale jejichž samotné prvky funkce nejsou. Základní kritéria pro tuto tvorbu určuje právě intenzionální báze.

4.2.1. Objektová báze

Jak je výše uvedeno jedná se o soubor množin. Příslušným intuitivním pojmům přiřazujeme právě jednotlivé objekty, které jsou chápány jako nulární funkce bez argumentů nebo funkce pomocí objektů vybudované. V TIL rozlišujeme čtyři různé prvky v bázi.

a) Pravdivostní hodnoty – tento prvek je množina dvou hodnot, označovaných jako P a N, jež vyjadřují Pravdu a Nepravdu. Tato báze je nutná, abychom vůbec mohli jakýmkoli výrazům přiřazovat pravdivostní hodnotu, tudíž abychom o jakékoli větě mohli tvrdit, že je nebo není pravdivá. Dále z důvodu parciality TIL víme, že ne vždy má smysl tvrdit, že je věta pravdivá, ale ani nepravdivá a taková věta tudíž nemá žádnou pravdivostní hodnotu.
b) Reálná čísla, také vyjadřující časové hodnoty – z důvodu potřeby zavedení čísel a rozeznání času, je tato množina dalším prvkem báze. Množina obsahuje nekonečně mnoho prvků, kde každý prvek odpovídá jednomu časovému okamžiku.
c) Světy – množina všech možných světů je třetím prvkem objektové báze. Její potřeba existence vyplývá z principu antiaktualismu. Konkrétně jde o posloupnost vzájemně si neodporujících empirických faktů v čase.
d) Univerzum – univerzum obsahuje individua, která existují sama o sobě, bez nutnosti mít některé vlastnosti.

4.2.2. Intenzionální báze

Intenzionální báze je určena množinou kritérií pro tvorbu funkcí nad objektovou bází. Přesněji se jedná o zobrazení ze všech možných světů do zobrazení všech možných časů do množiny objektů nad objektovou bází, tzv. typ intenze.

4.2.3. Typy

Pro analýzu přirozeného jazyka obvykle volíme, výše uvedenou, objektovou bázi, která se tomuto účelu jeví nejvhodnější.

a) Množina pravdivostních hodnot – množinu {P, N} typicky označujeme řeckým písmenem o.
b) Množina časových okamžiků a reálných hodnot – označujeme písmenem τ.
c) Množina světů – této množině bylo přiřazeno písmeno ω.
d) Univerzum – množina individuí je značena pomocí ι.

Prvky množin zvolené báze nazýváme atomické typy. Bázi můžeme volit podle problému, který řešíme nebo analyzujeme, proto například pro analýzu matematických faktů nepotřebujeme množinu světů ani množinu různých časů, a naopak můžeme zavést množinu přirozených čísel. Další, takzvané molekulární typy, můžeme vytvořit z těchto atomických typů. Molekulární typy jsou zobrazením kartézského součinu typů $\beta_1 \times \ldots \times \beta_n$ (kde n > 0) do typu α, v rámci TIL zapíšeme takto: $(\alpha\beta_1 \ldots \beta_n)$. Atomické i molekulární typy nazýváme typy řádu 1.

Při analýze přirozeného jazyka narazíme na výrazy, kterým není možné přiřadit pouze typy prvního řádu. Typickým případem může být věta "Petr řeší rovnici $x^2+1=0$". V tomto případě má individuum Petr vztah k výrazu rovnice, kterou řeší. V TIL má tedy vztah k celé konstrukci, jejíž typ neumíme vyjádřit typem prvního řádu. Pokud by to bylo možné a přiřadili bychom rovnici molekulární typ prvního řádu, vyjadřoval by typ Petrův vztah k výrazu, což by způsobilo, že řeší zároveň všechny

ekvivalentní rovnice, o kterých ale nemusí vědět. Pokud by rovnice měla atomický typ, měl by Petr vztah k výsledku, tedy by již v tu chvíli věděl výsledek rovnice, a řešit ji by nedávalo smysl.

Typ, který označuje takovouto konstrukci označujeme obecně $*_n$, kde n + 1 je řádem typu. V našem příkladu, by tedy rovnice měla typ $*_1$ řádu 2. Konstrukce konstruující objekt s typem řádu 2, bude mít typ řádu 3, a tak dále. S přidáním typů vyšších řádů mluvíme o rozvětvené hierarchii typů nad zvolenou bází.

Dále rozlišujeme typy na takzvané α-intenze, které označují typy závislé na množině světů, a tedy je používáme k analýze empirických výrazů a α-extenze, které jsou na množině světů nezávislé. Empirické výrazy jsou takové, které nabývají různých hodnot alespoň ve dvou různých světech nebo časech, ostatní výrazy jsou analytické. U empirických výrazů je rozdíl denotátem, který označujeme intenzí, a referencí, která je hodnota intenze v určitém světě a čase. Analytické výrazy mohou být i bez denotátu.

Příklady některých objektů a jejich typů:

- Množiny – množiny mají typ (oα).

Příklad: Množina čísel menších než 10.

$$\lambda x[{}^0< x\ {}^0 10] \rightarrow (o\tau)$$

Příklad: Množina binárních relací na číslech 1 a 2.

$$\lambda x[x\ {}^0 1\ {}^0 2] \rightarrow (o(o\tau\tau)), \text{ kde } x \rightarrow (o\tau\tau)$$

- Operace – operace mají typ $(\alpha\beta_1 \ldots \beta_n)$, kde α je návratový typ, β_i je typ hodnoty, na kterou je operace aplikována, a n je arita.

*Příklad: **Součet** čtverky a pětky.*

$$[{}^0+\ {}^0 4\ {}^0 5], \text{ kde } +/(\tau\tau\tau)$$

*Příklad: **Sjednocení tří množin** čísel, kde první množina obsahuje hodnoty větší než sto, druhá množina mezi deseti a sty a třetí množina nabývá hodnot menších než deset.*

$$[{}^0\text{Sjednocení\_tří\_množin}\ \lambda x[{}^0\geq x\ {}^0 100]\ \lambda y[[{}^0< y\ {}^0 100] \wedge [{}^0\geq y\ {}^0 10]]\ \lambda z[{}^0< z\ {}^0 10]], \text{ kde Sjednocení\_tří\_množin}/((o\tau)(o\tau)(o\tau)(o\tau))$$

- Individuové úřady – nebo také role, mají typ $((\iota\tau)\omega)$, obvykle zkracujeme na $\iota_{\tau\omega}$. Tento typ mají výrazy, které označují úřady nebo role, které může zastávat jedno individuum, ale ono individuum se může měnit v různých časech nebo světech.

Příklad: Král Anglie.

$$\lambda w \lambda t[{}^0\text{Král}_{wt}\ {}'{}^0\text{Anglie}] \rightarrow \iota_{\tau\omega}, \text{Král}/(\iota)_{\tau\omega}$$

Protože se například král Anglie mění v průběhu času, nemůžeme krále Anglie analyzovat jako individuum, protože by to buď znamenalo, že dané individuum bylo a bude králem Anglie vždy a ve všech možných světech nebo, že jsou všichni králové Anglie jedno a to samé individuum.

- Vlastnosti individuí – typu $(((o\iota)\tau)\omega)$ nebo $(o\iota)_{\tau\omega}$. Tento typ mají výrazy, které označují empirickou množinu individuí, která se může měnit v různých časech a světech.

*Příklad: Někdo je **opilý**.*

$$\lambda w \lambda t[{}^0\exists^\iota\ \lambda x[{}^0\text{Opilý}_{wt}\ x]], \text{ kde Opilý}/(o\iota)_{\tau\omega},\ \exists^\iota/(o(o\iota))$$

- Atributy – také možno nazývat empirické funkce mají typ $(\alpha\beta)_{\tau\omega}$, kde α není o, typicky $(\iota)_{\tau\omega}$. Přiřazují určitému prvku β jiný prvek α, v případě $(\iota)_{\tau\omega}$ přiřazuje atribut danému individuu nejvýše jedno individuum. Stejně jako role a vlastnosti, atributy jsou empirické typy, které jsou závislé na množině časů a světů.

Příklad: ***Manželka*** *Karla.*

$\lambda w \lambda t[{}^0\text{Manželka}_{wt}\ {}^0\text{Karel}]$, kde Manželka/$(\iota)_{\tau\omega}$, Karel/$\iota$

4.2.4. Konstrukce

Konstrukce jsou ve své podstatě abstraktním popisem konkrétních procedur. Konkrétní procedura je například program, který realizuje určitý algoritmus, a algoritmus je abstraktní popis programu. Konkrétní procedura je navíc jasně vymezitelná v čase a prostoru. Abstraktní procedura je v TIL pokládána za význam jazykových výrazů. Pro zápis konstrukcí používáme v TIL modifikovanou variantu typovaného lambda kalkulu, který je parciální a hyperintenzionální.

Zavedením parciality se vytváří určité problémy, ale i možnosti. Můžeme například zapsat výrazy, které dávají smysl, ale neexistují. Například výraz *nejmenší celé číslo* dává smysl a chápeme význam tohoto výrazu, ale jelikož takové číslo neexistuje, tak konstrukce, která popisuje tento výraz nebude vracet žádnou hodnotu.

Další důležitou vlastností TILu je, že termy jazyka konstrukcí nevyžadují interpretaci pro to, aby nabyly významu, jak tomu běžně bývá v jiných, často jednodušších logikách. Toto je možné, protože termy označují přímo konstrukce, a to přesně definicí daným způsobem. Díky tomu nazýváme TIL hyperintenzionální kalkul, protože oproti intenzionálním kalkulům, kde je význam termů funkce konstruovaná procedurou zakódovanou termem, v TIL je významem procedura samotná. Mimo jiné to znamená, že v TIL rozlišujeme mezi objektem a jeho konstrukcí a je možné, že tento objekt může být i jiná konstrukce. Konstrukce, která operuje na jiné konstrukci má vyšší řád. Konstrukce, které jsou součástí jiných konstrukcí nazýváme podkonstrukce nebo také konstituenty.

Abychom mohli dodat objekty pro procedury konstrukcím, je nutno definovat dvě základní, tzv. atomické konstrukce. Tyto konstrukce se nazývají proměnná a trivializace a nemají jiné konstituenty než samy sebe. Další, již molekulární, konstrukce, které mají i jiné konstituenty než samy sebe, neboli obsahují podkonstrukce, jsou kompozice, uzávěr, provedení a dvojí provedení.

- Proměnné – jedná se o atomickou konstrukci, která konstruuje objekty na základě valuace *v*. Proto říkáme, že proměnná *v*-konstruuje objekt příslušného typu *α*. *V*-konstrukce pro typ *α* funguje asi takto: pro spočetně nekonečnou množinu proměnných uspořádáme prvky typu α do spočetně nekonečně mnoho posloupností a na základě parametru v určíme posloupnost, která přiřadí všem proměnným daný prvek.

Příklad: $x + y$

$[{}^0+\ x\ y]$.

Kde x a y jsou proměnné, které jsou sčítány.

- Trivializace – tato atomická konstrukce vrací objekty pro ostatní konstrukce takovým způsobem, že trivializace objektu X s nějakým typem α prostě konstruuje tento daný objekt bez jakýchkoli úprav. Na typu *α* nezáleží, trivializovány mohou být i samotné konstrukce. Pokud máme objekt X, zapíšeme trivializaci jako ${}^0X$. Potřeba trivializace vychází už z toho,

že objekt je v TILu něco jiného než konstrukce a nemůže konstruovat sám sebe, potřebujeme konstrukci, která ho bude konstruovat. Dalším využitím trivializace je možnost trivializovat samotné konstrukce, čímž můžeme zmiňovat konkrétní procedury, a ne pouze jejich hodnotu. Nad trivializací objektu můžeme také uvažovat jako na ukazatel na daný objekt.

Příklady: $^0 1$, $^0 Tom$, $^0+$, $^0 Prezident$, $^0 Hledá$.

Typy: $1/\tau$, Tom/ι, $Prezident/(\iota)_{\tau\omega}$, $Hledá/(o\iota_{\tau\omega})_{\tau\omega}$

- Kompozice – kompozice je konstrukce vycházející z operace aplikace v lambda kalkulu. Jde o provedení funkce f s typem $(\alpha\beta_1 \ldots \beta_n)$ na daných argumentech $A_1 \ldots A_n$ s typy $\beta_1 \ldots \beta_n$, abychom na těchto argumentech dostali hodnotu. Protože je TIL parciální systém, je možné, že pro danou množinu argumentů nebude existovat žádná hodnota a kompozice tedy nebude vracet žádnou hodnotu. Pokud konstrukce nevrací žádnou hodnotu pro danou valuaci říkáme, že je v-nevlastní. Kompozice má podobu $[X\, Y_1 \ldots Y_n]$, kde X konstruuje funkci f a konstrukce Y_i konstruuje i-tý argument funkce.

Příklad: 1 + 2 = 3

$[^0= [^0+ \, ^0 1 \, ^0 2] \, ^0 3]$

Příklad: Josef hledá vedoucího katedry.

$\lambda w \lambda t [^0 Hledat_{wt} \, ^0 Josef \, \lambda w_1 \lambda t_1 [^0 Vedoucí_{wt} \, ^0 Katedra]]$

- Uzávěr – další konstrukce vycházející z lambda kalkulu, tentokrát z operace abstrakce. Uzávěr konstruuje funkci f tím, že abstrahuje hodnoty argumentů. Jedná se tedy o duální operaci ke kompozici. Uzávěr zapisujeme $[\lambda x_1 \ldots x_n \, Y]$, kde $x_1 \ldots x_n$ jsou od sebe různé proměnné s typy $\beta_1 \ldots \beta_n$ a Y je konstrukce, která konstruuje objekt s typem α, výsledný typ funkce f bude $(\alpha\beta_1 \ldots \beta_n)$. Uzávěr vždy konstruuje funkci f, ale funkce f může být v-nevlastní a tedy sama nebude nikdy nic vracet.

Příklad: Funkce následovníka. (Přičte k číslu jedna.)

$\lambda x [^0+ \, x \, ^0 1]$

- Provedení – konstrukce zapsaná $^1 X$ pro konstrukci X, pouze konstruuje objekt konstruovaný konstrukcí X nebo nekonstruuje nic.
- Dvojí provedení – tato konstrukce je podobná provedení, ale dá se použít na konstrukce vyšších řádů a udělat provedení dvakrát. Použití na konstrukci X zapíšeme $^2 X$. Dvojí provedení je v-nevlastní, pokud X není konstrukce nebo nekonstruuje konstrukci Y nebo konstrukce Y nekonstruuje objekt. Dvojí provedení můžeme použít pro získání hodnoty z hyperintenzionální konstrukce nebo konstrukce vyššího řádu.

Příklad: Jan hledal nejrychlejšího člověka a našel ho.

$\lambda w \lambda t [[^0 Hledat_{wt} \, ^0 Jan \, \lambda w_1 \lambda t_1 [^0 Nejrychlejší_{wt} \, ^0 Člověk]] \wedge {}^2 [^0 Sub \, ^{00} Jan \, ^0 x \, ^0 [\lambda w \lambda t [^0 Najít_{wt} \, x]]]]$

Podkonstrukce a konstituenty

Tato podkapitola vychází ze zdroje 1.

Konstrukce mohou obsahovat jiné podkonstrukce a tzv. konstituenty. Definice podkonstrukce pro konstrukci *C* je poměrně přímočará;

1. *C* je sama o sobě vlastní podkonstrukcí.
2. Pokud je *C* trivializací, provedením nebo dvojím provedením jiné konstrukce *X*, pak konstrukce X je podkonstrukcí C.
3. Je-li *C* kompozice $[X\ Y_1 \ldots Y_n]$, pak jsou všechny části kompozice $X, Y_1, \ldots, Y_n$ také podkonstrukce *C*.
4. Pokud je *C* uzávěr, poté konstrukce, na kterou je uzávěr aplikován je také podkonstrukce. Tedy pro $C = [\lambda x_1 \ldots x_n\ Y]$ je *Y* podkonstrukce.
5. Relace být podkonstrukcí je tranzitivní, tudíž pokud má *C* podkonstrukci *B* a ta má podkonstrukci *A*, pak *A* je podkonstrukcí *C*.
6. Nic jiného není podkonstrukcí, než co je popsáno v bodech 1-5.

Konstituenty jsou specifické podkonstrukce, které splňují tu vlastnost, že nejsou pouze zmiňovány, ale jsou užity a musí být užity pro to, aby bylo možno provést konstrukci C, které jsou součástí. Pokud máme podkonstrukce, které nejsou konstituenty, jsou pouze zmiňovány, vyskytují se hyperintenzionálně a jsou použity jinými funkcemi.

Kvantifikátory

V TIL je zavedených mnoho specifických funkcí, mezi nimiž jsou i kvantifikátory. Tyto specifické funkce zde uvádím, protože se často používají při analýze výrazů a s kvantifikátory pracuje několik pravidel přirozené dedukce, které umí program používat.

Protože vycházíme i z odvozovacích pravidel přirozené dedukce v PL1, potřebujeme zavést co jsou v TIL kvantifikátory a jak fungují. V TIL oproti PL1 musí mít kvantifikátory význam sám o sobě a musí být objektem jako všechno ostatní. Ekvivalentem kvantifikátorů z PL1 jsou kvantifikátory neomezené a jedná se o funkce určitého typu. Máme kvantifikátor všeobecný $\forall^{\alpha}$ a existenční $\exists^{\alpha}$ s typem $(o(o\alpha))$, kde je typ α určen podle typu množiny, na kterou je kvantifikátor aplikován. Neomezených kvantifikátorů je tedy defacto nekonečně mnoho, ale jsou až na typ prakticky stejné. Všeobecný kvantifikátor vrací hodnotu P (pravda), pokud množina, na kterou je aplikován obsahuje všechny prvky typu α. Existenční kvantifikátor vrací P, pokud množina obsahuje alespoň jeden prvek. Protože se kvantifikátory často používají zkracujeme jejich zápis tak, že nepíšeme trivializaci kvantifikátoru, jeho typ a lambda znak uzávěru na který je aplikován.

Příklad: Všichni učitelé jsou chytří.

Celý zápis: $[{}^0\forall^{\iota}\lambda x\,[[{}^0\text{Učitel}_{wt}\ x] \supset [{}^0\text{Chytrý}_{wt}\ x]]]$

Zkrácený zápis: $[\forall x\,[[{}^0\text{Učitel}_{wt}\ x] \supset [{}^0\text{Chytrý}_{wt}\ x]]]$

Příklad: Existuje číslo menší než 0.

Celý zápis: $[{}^0\exists^{\tau}\lambda x\,[{}^0< x\,{}^00]]$

Zkrácený zápis: $[\exists x\,[{}^0< x\,{}^00]]$

4.2.5. Kontexty

Podkonstrukce v TIL se mohou vyskytovat ve dvou různých režimech – mohou být užity nebo zmíněny. Určitý výraz je zmíněn, pokud je sám použit jako objekt a můžeme o něm něco vypovědět. Pokud je výraz zmíněn, pak je to díky jinému výrazu, který je užit. Příkladem může být: "Výrazem "auto" myslíme určitý druh dopravního prostředku.", kde je výraz "auto" užit jako objekt a něco o něm vypovídáme. Výrazy, které pouze zmiňujeme se vyskytují hyperintenzionálně. Pokud je výraz naopak užit a je tedy konstituentem konstrukce, pak je použit k tomu, aby nám po provedení dal nějaký výsledek. Způsob užití dále rozdělujeme podle toho, jestli je funkce vytvořená konstrukcí výrazu užita jako argument nebo je použita její hodnota. Pokud použijeme hodnotu funkce na určitých argumentech jako další hodnoty predikace (obvykle v kompozici), říkáme, že se výraz vyskytuje extenzionálně. Pokud je ovšem celá funkce použita jako argument pro jinou funkci mluvíme o kontextu intenzionálním. Konstrukce atomického typu musí být v intenzionálním kontextu, protože její funkce nemůže přijímat žádné argumenty. Rozdíly mezi zmiňováním a užíváním a užitím hodnoty nebo celé funkce nám dávají tři kontexty.

- Hyperintenzionální kontext – v tomto kontextu není konstrukce užita, aby byla konstituentem, vracela hodnotu nebo svůj význam, ale je pouze zmíněna a použita jako objekt, který může být argumentem funkce.

  Příklad: ***Ovce*** *je slovo.*

  $[{}^{0}\text{Slovo}_{wt}\ {}^{00}\text{Ovce}]$, Ovce/ι, Slovo/$(o^*)_{\tau\omega}$

- Intenzionální kontext – konstrukce je užita pro vytvoření funkce vyjadřující její význam a je dále použita celá funkce, ne její hodnota. Dále se konstrukce nevyskytuje v hyperintenzionálním kontextu.

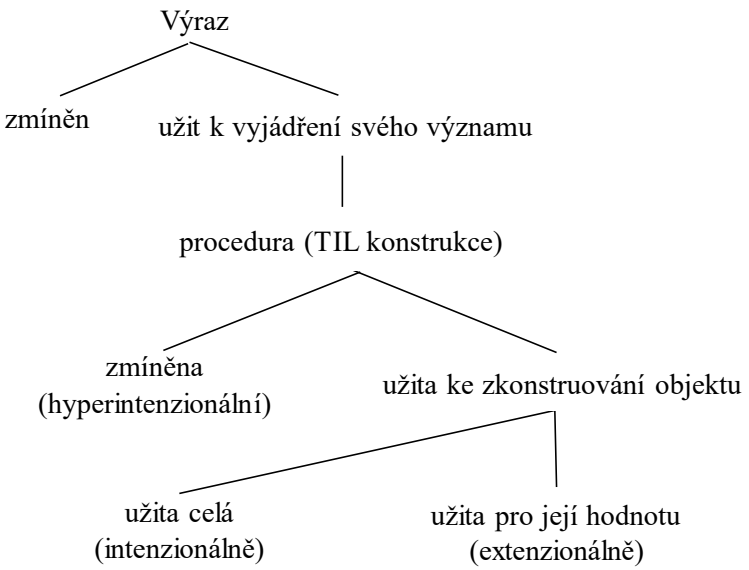
  Příklad: ***Karel*** *je prezident.*

  $[{}^{0}\text{Prezident}_{wt}\ {}^{0}\text{Karel}]$, Karel/ι, Prezident/$(o\iota)_{\tau\omega}$

- Extenzionální kontext – konstrukce je užita pro vytvoření funkce, na kterou jsou aplikovány argumenty a výsledná hodnota je objektem predikace. Dále není užitá konstrukce v intenzionálním nebo hyperintenzionálním kontextu.

  Příklad: Karel je ***prezident****.*

$[^0\text{Prezident}_{wt}\ ^0\text{Karel}]$, Karel/ι, Prezident/$(o\iota)_{\tau\omega}$



Obr. 2 - Schéma vyjadřující vztah zmiňování a užívání výrazu a konstrukce k jejich kontextu

Některé kontexty mají přednost před jinými. Nejvyšší kontext je hyperintenzionální a pokud se podkonstrukce vyskytuje hyperintenzionálně nemůže se vyskytovat v jiném kontextu. Na další úrovni je kontext intenzionální a až poté extenzionální. Proto se může stát, že pokud podkonstrukce, která by měla mít extenzionální kontext a je v dosahu intenzionálního uzávěru, pak se bude vyskytovat intenzionálně. Pokud tedy máme například konstrukci [λx ['+ x '1]], pak se funkce součtu vyskytuje intenzionálně, protože je v dosahu intenzionálního uzávěru λx.

Příklad: Petr řeší rovnici $x^2 + 1$.

$$[\lambda w\ [\lambda t\ [[[^0\text{Řešit}\ w]\ t]\ ^0\text{Petr}\ ^0[^0+\ [^0\text{Mocnina}\ x\ ^02]\ ^01]]]]$$

1. $[\lambda w\ [\lambda t\ [[[^0\text{Řešit}\ w]\ t]\ ^0\text{Petr}\ ^0[^0+\ [^0\text{Mocnina}\ x\ ^02]\ ^01]]]]$ – výskyt intenzionální
2. $[\lambda t\ [[[^0\text{Řešit}\ w]\ t]\ ^0\text{Petr}\ ^0[^0+\ [^0\text{Mocnina}\ x\ ^02]\ ^01]]]$ – výskyt intenzionální
3. $[[[^0\text{Řešit}\ w]\ t]\ ^0\text{Petr}\ ^0[^0+\ [^0\text{Mocnina}\ x\ ^02]\ ^01]]$ – výskyt intenzionální
4. $[[^0\text{Řešit}\ w]\ t]$ – výskyt intenzionální v celé konstrukci, byl by extenzionální v 3
5. $[^0\text{Řešit}\ w]$ - výskyt intenzionální v celé konstrukci, byl by extenzionální v 3
6. ^{0}Řešit - výskyt intenzionální v celé konstrukci, byl by extenzionální v 3
7. w, t – výskyt intenzionální
8. ^{0}Petr – výskyt intenzionální
9. $^0[^0+\ [^0\text{Mocnina}\ x\ ^02]\ ^01]$ – výskyt hyperintenzionální
10. $[^0+\ [^0\text{Mocnina}\ x\ ^02]\ ^01]$ – výskyt hyperintenzionální
11. $^0+$ – výskyt hyperintenzionální, extenzionální v 10
12. $[^0\text{Mocnina}\ x\ ^02]$ – výskyt hyperintenzionální, intenzionální v 10
13. ^{0}Mocnina – výskyt hyperintenzionální, extenzionální v 12
14. x, ^{02} – výskyt hyperintenzionální, intenzionální v 12
15. ^{01} – výskyt hyperintenzionální, intenzionální v 10

Abychom mohli správně usuzovat v TIL pomocí pravidel přirozené dedukce, musí brát pravidla kontext v potaz. Některým pravidlům na kontextu nezáleží, některá pravidla jako je existenční generalizace se

u různých kontextů mírně liší a jiná pravidla jako beta redukce jménem nemusí být v některých případech validní, kvůli určitým kontextům.

5. TIL-Script

Tato kapitola vychází ze zdroje 4.

TIL-Script je komputační verze transparentní intenzionální logiky. TIL-Script zahrnuje veškerou funkcionalitu TIL. Od svého vzoru se liší především jednodušším zápisem, ale také upravuje a rozšiřuje množinu atomických typů. Jednodušší zápis umožňuje přehledné vypsání výrazů na počítači bez složitého formátování a používá pouze znaky ze sady ASCII. Řecká písmena, speciální znaky a kvantifikátory jsou nahrazeny klíčovými slovy. Typ pro časový údaj a reálné číslo byl rozdělen na dva samostatné typy, dále přibyly typy pro přirozená čísla a typ pro řetězec znaků. Naopak typ $*_n$ je zjednodušen pouze na *, tudíž neznačíme řád typu v rozšířené hierarchii typů. U uzávěru můžeme definovat typ proměnné v definici uzávěru za dvojtečku nebo zapsat typ proměnné podobně jako typ objektu.

TIL	TIL-Script	Popis
ο	Bool	Pravdivostní hodnota
ι	Indiv	Individuum
τ	Time	Čas
τ	Real	Reálné číslo
ω	World	Svět
α	Any	Jakýkoliv typ
$*_n$	Star, *	Typy vyššího řádu
-	String	Řetězec
-	Int	Celé číslo

Tabulka 1 - Seznam typů v TIL a jejich ekvivalenty v TIL-Scriptu

TIL	TIL typ	TIL-Script	TIL-Script typ	Popis
∧	(ooo)	And	(Bool Bool Bool)	Konjunkce
∨	(ooo)	Or	(Bool Bool Bool)	Disjunkce
⊃	(ooo)	Implies	(Bool Bool Bool)	Implikace
≡	(ooo)	Equiv	(Bool Bool Bool)	Ekvivalence
¬	(oo)	Not	(Bool Bool)	Negace
∃	(o(oα))	Exist	(Bool (Bool Any))	Existenční kvantifikátor
∀	(o(oα))	ForAll	(Bool (Bool Any))	Všeobecný kvantifikátor

Tabulka 2 - Seznam základních operací a kvantifikátorů v TIL a TIL-Scriptu a jejich typů

Příklad zápisu v TIL a TIL-Scriptu:

Všichni studenti jsou chytří.

TIL:

$$\lambda w \lambda t[\forall x[[{}^0\text{Student}_{wt}\ x] \supset [{}^0\text{Chytrý}_{wt}\ x]]]$$

$$w \rightarrow \omega;\ t \rightarrow \tau;\ x \rightarrow \iota;\ \forall/(o(o\alpha));\ \text{Student,Chytrý}/(o\iota)_{\tau\omega};\ \supset/(ooo).$$

TIL-Script:

```
[\w [\t ['ForAll [\x ['Implies ['Student@wt x] ['Chytrý@wt x]]]]]].
```

```
x -> Indiv, w -> World, t -> Time, ForAll/(Bool (Bool Any)), Implies/(Bool Bool Bool),
Student,Chytrý/(((Bool Indiv) Time) World).
```

6. Přirozená dedukce v TIL-Scriptu

Tato kapitola vychází ze zdroje 4.

6.1. Pravidla přirozené dedukce

6.1.1. Zavedení konjunkce

$$A, B \vdash [{}'And\ A\ B].$$

Příklad: V Ostravě je deštivo. V Brně je větrno. ⊢ V Ostravě je deštivo a v Brně je větrno.

['Deštivo@wt 'Ostrava]. ['Větrno@wt 'Brno]. ⊢
['And ['Deštivo@wt 'Ostrava] ['Větrno@wt 'Brno]].

Typy: Ostrava, Brno/Indiv, Deštivo, Větrno/(Bool Indiv)@tw.

6.1.2. Eliminace konjunkce

$$[{}'And\ A\ B] \vdash A, B$$

Příklad: V Ostravě je deštivo a v Brně je větrno. ⊢ V Ostravě je deštivo. V Brně je větrno.

['And ['Deštivo@wt 'Ostrava] ['Větrno@wt 'Brno]]. ⊢
['Deštivo@wt 'Ostrava]. ['Větrno@wt 'Brno].

6.1.3. Zavedení disjunkce

$$A \vdash [{}'OrA\ B].$$

Příklad: V Ostravě je deštivo. ⊢ V Ostravě je deštivo nebo v Brně je větrno.

['Deštivo@wt 'Ostrava]. ⊢
['Or ['Deštivo@wt 'Ostrava] ['Větrno@wt 'Brno]].

6.1.4. Eliminace disjunkce

$$[{}'Or\ A\ B].[{}'Not\ B] \vdash A$$

$$[{}'Or\ A\ [{}'Not\ B]].B \vdash A$$

Příklad: V Ostravě je deštivo nebo v Brně je větrno. V Brně není větrno. ⊢ V Ostravě je deštivo.

['Or ['Deštivo@wt 'Ostrava] ['Vetrno@wt 'Brno]]. ['Not ['Vetrno@wt 'Brno]]. ⊢
['Deštivo@wt 'Ostrava].

6.1.5. Zavedení implikace

$$A \vdash [{}'Implies\ B\ A].$$

Příklad: V Ostravě je deštivo. ⊢ Když v Brně je větrno, tak je v Ostravě deštivo.

['Deštivo@wt 'Ostrava]. ⊢ ['Implies ['Větrno@wt 'Brno] ['Deštivo@wt 'Ostrava]].

6.1.6. Eliminace implikace (modus ponens)

$$[{}'Implies\ A\ B].A \vdash B$$

Příklad: *Když je v Ostravě deštivo, tak je v Brně větrno. V Ostravě je deštivo. ⊢ V Brně je větrno.*

['Implies ['Deštivo@wt 'Ostrava] ['Větrno@wt 'Brno]]. ['Deštivo@wt 'Ostrava]. ⊢
['Větrno@wt 'Brno].

6.1.7. Zavedení ekvivalence

$$[{'Implies}\ A\ B].[{'Implies}\ B\ A] \vdash [{'Equiv}\ A\ B].$$

Příklad: Když je v Ostravě deštivo, pak je v Brně větrno. Když je v Brně větrno, pak je v Ostravě deštivo. ⊢ V Ostravě je deštivo právě tehdy, když je v Brně větrno.

['Implies ['Deštivo@wt 'Ostrava] ['Větrno@wt 'Brno]]. ['Implies ['Větrno@wt 'Brno] ['Deštivo@wt 'Ostrava]]. ⊢ ['Equiv ['Deštivo@wt 'Ostrava] ['Větrno@wt 'Brno]].

6.1.8. Eliminace ekvivalence

$$[{'Equiv}\ A\ B]. \vdash [{'Implies}\ A\ B].[{'Implies}\ B\ A].$$

Příklad: *V Ostravě je deštivo právě tehdy, kdy je ž v Brně větrno. ⊢ Když je v Ostravě deštivo, pak je v Brně větrno. Když je v Brně větrno, pak je v Ostravě deštivo.*

['Equiv ['Deštivo@wt 'Ostrava] ['Větrno@wt 'Brno]]. ⊢ ['Implies ['Deštivo@wt 'Ostrava] ['Větrno@wt 'Brno]]. ['Implies ['Větrno@wt 'Brno] ['Deštivo@wt 'Ostrava]].

6.1.9. Komutativní pravidlo

Toto pravidlo je dokazatelné v systému přirozené dedukce, ale pro zjednodušení bylo přidáno jako samotné pravidlo.

$$[O\ A\ B] \vdash [O\ B\ A]. Kde\ O\ je\ operace\ {'And},\ {'Or}\ nebo\ {'Equiv}.$$

6.1.10. Zavedení existenčního kvantifikátoru (existenční generalizace)

Při provádění tohoto pravidla si musíme dát pozor na kontext prvku, nad kterým zavádíme existenční kvantifikátor. Kontext totiž určuje typ prvku, který kvantifikujeme. V hyperintenzionálním kontextu kvantifikujeme konstrukci, v intenzionálním kontextu jde o funkci, a v extenzionálním kontextu je typem hodnota funkce.

Hyperintenzionální verze (konstrukce K má hyperintenzionální kontext)

$$[\ldots {'}[\ldots {'}K\ \ldots]\ \ldots] \vdash [{'Exist}\ [\backslash x\ [\ldots [{'Sub}\ [{'Tr}\ x]\ {'y}\ {'}[\ldots y\ \ldots]]\ \ldots]], x \to *.$$

Příklad: Petr řeší rovnici $x^2 + 1$. ⊢ Existuje číslo y, pro které Petr řeší rovnici $x^2 + y$.

['Řešit@wt 'Petr '['+ ['Mocnina x '2] '1]]. ⊢ ['Exist [\y ['Řešit@wt 'Petr ['Sub ['Tr y] 'z '['+ ['Mocnina x '2] z]]]]].

Typy: 'Řešit/(Bool Indiv Star)@tw, Petr/Indiv, 1,2/Real, Mocnina,+/(Real Real Real), x → Real, y, '['+ ['Mocnina x '2] '1]] → Star.

Intenzionální verze (K má intenzionální kontext)

$$[\ldots K\ \ldots] \vdash [{'Exist}\ [\backslash x\ [\ldots x\ \ldots]], x \to (\alpha\ \beta_1\ \ldots\ \beta_n).$$

Příklad: Jan zjišťuje kdo je prezidentem České republiky (ČR). ⊢ Jan něco zjišťuje. (Jedná se o čtení de dicto)

['Zjišťovat@wt 'Jan [\w\t ['Prezident@wt 'ČR]]]. ⊢ ['Exist [\x ['Zjišťovat@wt 'Jan x]]].

Typy: Zjišťovat/(Bool Indiv Indiv@tw)@tw, Jan,ČR/Indiv, Prezident/(Indiv Indiv)@tw, w → World, t → Time, x → Indiv@tw.

Extenzionální verze (K má extenzionální kontext)

$$[\ldots K \ldots] \vdash [{}'Exist\ [\backslash x\ [{}'=\ x\ K]]], K \rightarrow (\alpha\ \beta_1 \ldots\ \beta_n), x \rightarrow\ \alpha.$$

Příklad: Premiér Anglie je politik. ⊢ Premiér Anglie existuje.

['Politik@wt [\w [\t ['Premiér 'Anglie]]]@wt]. ⊢ ['Exist [\x ['= x [[\w\t ['Premiér 'Anglie]]@wt]]]].

Typy: Politik/(Bool Indiv)@tw, Anglie/Indiv, Premiér/(Indiv Indiv), w → World, t → Time, x → Indiv.

6.1.11. Eliminace existenčního kvantifikátoru

Jelikož toto pravidlo nezachovává pravdivost, ale pouze splnitelnost, můžeme ho samostatně použít pouze v případě nepřímého důkazu. Pokud chceme pravidlo použít pro přímý důkaz, musíme existenční kvantifikátor opět zavést, abychom zachovali pravdivost. Pokud je existenční kvantifikátor v dosahu všeobecného kvantifikátoru, musíme proměnnou nahradit novým funkčním symbolem, jehož hodnota je závislá na proměnné všeobecného kvantifikátoru.

Samostatný existenční kvantifikátor

$$[{}'Exist\ [\backslash x\ [K \ldots x \ldots]]]. \vdash [K \ldots {}'a \ldots].$$

Příklad: Existuje člověk, který je rozumný. ⊢ Tom je člověk a je rozumný.

['Exist [\x ['And ['Člověk@wt x] ['Rozumný@wt x]]]]. ⊢ ['And ['Člověk@wt 'Tom ['Rozumný@wt 'Tom]].

x → Indiv, Člověk,Rozumný/(Bool Indiv)@tw, Tom/Indiv.

Existenční kvantifikátor za všeobecným kvantifikátorem

$$[{}'ForAll\ [\backslash y\ [{}'Exist\ [\backslash x\ [\ldots\ x \ldots\ y \ldots]]]]]. \vdash [{}'ForAll\ [\backslash y\ [\ldots [{}'F\ y]\ \ldots\ y \ldots]]].$$

Příklad: Pro každé reálné číslo existuje o jedno menší číslo. ⊢ Každé reálné číslo se rovná výsledku funkce F(y).

['ForAll [\y ['Exist [\x ['= ['Minus y '1] x]]]]]. ⊢ ['ForAll [\y ['= ['Minus y '1] ['F y]]]].

x,y → Real, =/(Bool Real Real), Minus/(Real Real Real), 1/Real, F/(Real Real)

6.1.12. Zavedení všeobecného kvantifikátoru

Podobně jako u eliminace existenčního kvantifikátoru toto pravidlo samo o sobě nezachovává pravdivost. Z tohoto důvodu můžeme pravidlo validně použít pouze v případě, že jsme v předcházejících krocích důkazu použili eliminaci všeobecného kvantifikátoru.

$$\left[\left[\backslash x\ [\ldots x \ldots]\right] B\right]. \vdash [{}'ForAll\ [\backslash x\ [\ldots x \ldots]]].$$

Příklad: Petr je slušný. ⊢ Všichni jsou slušní.

[[\x ['Slušný@wt x]] 'Petr] ⊢ ['ForAll [\x ['Slušný@wt x]]].

x → Indiv, Slušný/(Bool Indiv)@tw, Petr/Indiv

6.1.13. Eliminace všeobecného kvantifikátoru

$$[{}'ForAll\ [\backslash x\ [\ldots x \ldots]]]. \vdash \left[\left[\backslash x\ [\ldots x \ldots]\right] B\right].$$

Příklad: Všichni jsou slušní. ⊢ Petr má vlastnost být slušný.

['ForAll [\x ['Slušný@wt x]]]. ⊢ [[\x ['Slušný@wt x]] 'Petr].

6.1.14. Beta konverze jménem

Beta konverze jménem je validní pouze v případě, že se její aplikací nezmění kontext konstrukcí.

$$[\backslash x\ [\ldots x \ldots x \ldots]\ N]. \vdash [\ldots N \ldots N \ldots].$$

Příklad: [[\x [\y ['Mul y x]] ['Div '1 '5]]]. ⊢ [\y ['Mul y ['Div '1 '5]]].

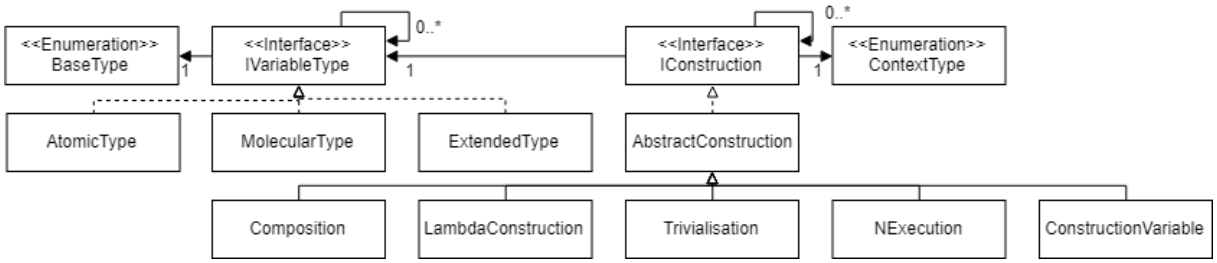
6.1.15. Beta konverze hodnotou

Oproti konverzi jménem je konverze hodnotou vždy validní.

$$[\backslash x\ [\ldots x \ldots x \ldots]\ N]. \vdash\ \hat{}2[\prime Sub\ [\prime Tr\ N]\ \prime x\ \prime[\ldots x \ldots x \ldots]].$$

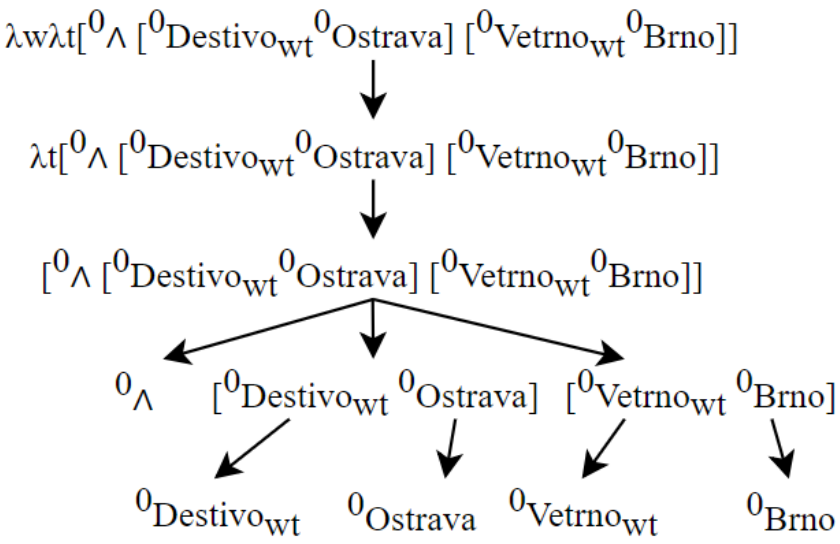
Příklad: [\x [\y ['Mul y x]] ['Div '1 '5]]. ⊢ ^2['Sub ['Tr ['Div '1 '5]] 'x '[\y ['Mul y x]]].

7. Analýza



Obr. 3 - Třídní diagram konstrukcí a jejích typů

7.1. Reprezentace konstrukcí



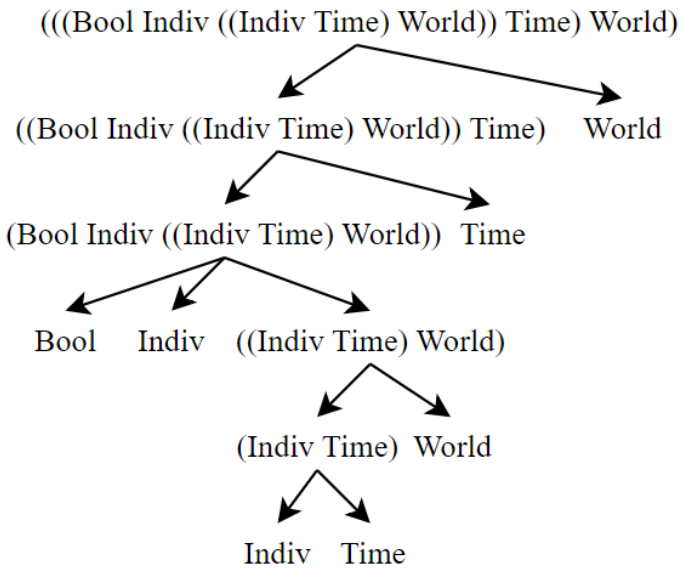
Obr. 4 - Strom podkonstrukcí konstrukce

Abychom mohli efektivně s konstrukcemi pracovat, potřebujeme je reprezentovat v programu vhodným datovým typem. Takový datový typ můžeme vytvořit, pokud se důkladně podíváme na definice různých typů konstrukcí. Z definice konstrukce kompozice vychází, že konstrukce se do sebe mohou vnořovat, a tedy, že konstrukce může obsahovat různé podkonstrukce a ty dále mohou obsahovat podkonstrukce. Takovouto strukturu si můžeme představit jako strom, viz obrázek výše. Všechny typy konstrukcí mohou být součástí tohoto stromu, a proto potřebujeme abstrahovat jejich obecné vlastnosti, abychom mohli se všemi typy konstrukcí pracovat jednotně, pokud to jde. Dále bychom chtěli, abychom mohli přistupovat k celému stromu stejně jako k samotnému prvku a nemuseli řešit konkrétní podobu dané konstrukce. Tyto problémy můžeme vyřešit použitím návrhového vzoru kompozit, který ukazuje, jak abstrahovat přístup ke stromu a jeho prvkům. Vytvoříme si obecný typ, který bude reprezentovat všechny konstrukce, a následně z něj budeme vytvářet konkrétní podoby jednotlivých typů konstrukcí.

- Proměnná – proměnná má oproti ostatním konstrukcím své jméno a přiřazený typ, který je buď striktně zadán u klíčových slov nebo vychází ze zadaných typů v kódu. Tento typ pro reprezentaci je použit jak pro proměnné, tak pro objekty, na které je následně aplikována trivializace.

- Trivializace – trivializace konstruuje objekt trivializace beze změny. Při trivializaci proměnné nebo kompozice mění typ na hvězdičku. Při aplikaci na kompozici se dále mění kontext kompozice na hyperintenzionální.
- Kompozice – jako jediná konstrukce rozvětvuje strom konstrukce, může mít tedy více potomků. Typ kompozice je dán výstupním typem prvního potomka.
- Uzávěr – uzávěr obsahuje jednu nebo více proměnných a může obsahovat i jejich typ, případně může být jejich typ dán jinde v kódu. Následné konstrukce, které jsou potomci ve stromu mohou obsahovat tyto proměnné. Typ uzávěru je molekulární typ ($\beta\alpha_1 \ldots \alpha_n$), jehož výstupní typ β je výstupním typem dítěte uzávěru ve stromu, a další typy jsou určeny typy proměnných α.
- Provedení – tento typ konstrukce není explicitně reprezentován.
- Dvojí provedení – v TIL-Scriptu a programu je tento typ konstrukce nahrazen N provedením, které se obvykle používá pouze jako dvojí provedení, ale je zobecněno.

7.2. Reprezentace typů

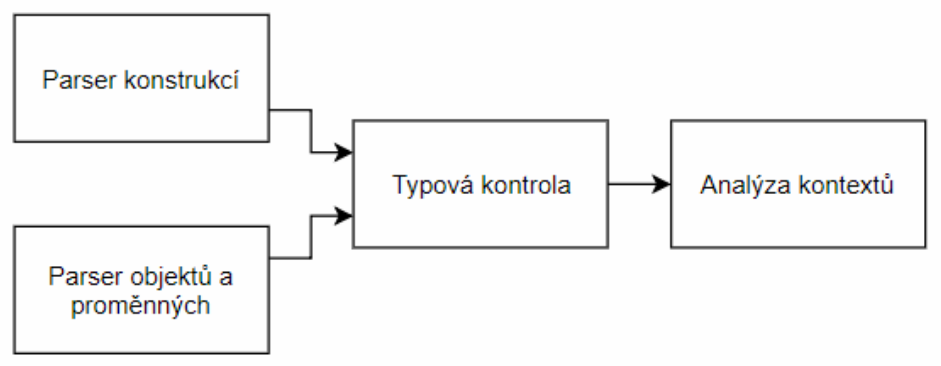


Obr. 5 - Strom podtypů v typu konstrukce

Z definice typů víme, že jsou typy atomické, molekulární a typy vyšších řádů. Jednoduché typy bychom mohli reprezentovat pouze výčtem možných hodnot, ale molekulární typy jsou množinou jiných typů, jak atomických, tak molekulárních. Tím, jako u reprezentace konstrukcí, narážíme na problém s vnořováním, který si opět můžeme představit stromem, a použijeme k vyřešení návrhový vzor kompozit. Pro zjednodušení programování budou všechny třídy typů obsahovat metody pro procházení stromu typu a další pomocné metody.

- Atomické typy – pro atomické typy nám stačí si uložit jméno z výčtu možných typů.
- Molekulární typy – zde musíme ukládat všechny typy, které jsou součástí tohoto molekulárního typu, případně i rodiče tohoto typu, abychom mohli procházet stromem typu.
- Typy vyšších řádů – v TIL-Scriptu nerozeznáváme různé řády typů, ale máme pouze typ Star nebo *. Většinou se tento typ chová jako atomický, ale můžeme chtít použít hodnotu, která je v něm schovaná po trivializaci konstrukce, například při N provedení. Proto si musíme pamatovat původní typ, který byl trivializován, čímž vznikl typ vyššího řádu.

7.3. Parsování konstrukcí



Obr. 6 - Průběh analýzy konstrukce

Abychom mohli převést text do naší datové struktury pro konstrukce potřebujeme je zparsovat. Parsování probíhá tak, že se kontrolují jednotlivé znaky věty v TIL-Scriptu a na základě nich se tvoří strom konstrukce. Podle specifických znaků můžeme určit konkrétní typy konstrukcí, které věta obsahuje.

- "[" - počátek kompozice, vytvoříme novou kompozici, která bude dítětem současné konstrukce, a následně se stane současnou konstrukcí, ve které se budeme pohybovat.
- "]" – konec kompozice, posuneme se ve stromu konstrukce zpět nahoru ke kompozici, která je rodičem současné konstrukce.
- "\" – počátek uzávěru, řetězec mezi tímto znakem a další konstrukcí jsou definice proměnných a typů daného uzávěru.
- "'" – apostrof značí trivializaci, kterou si vytvoříme a uložíme jako současnou konstrukci.
- "@" – tento znak značí, že je současná konstrukce závislá na světě a času.
- "." – tímto znakem ukončíme konstrukci a tím i parsování.

Po parsování se vytvořený strom konstrukce projde a najdou se všechny části závislé na světě a čase a rozšíří se do vlastních podkonstrukcí, například převedeme α@wt na [[α w] t], abychom si zjednodušili práci s analýzou typů a zvýšili přehlednost stromu.

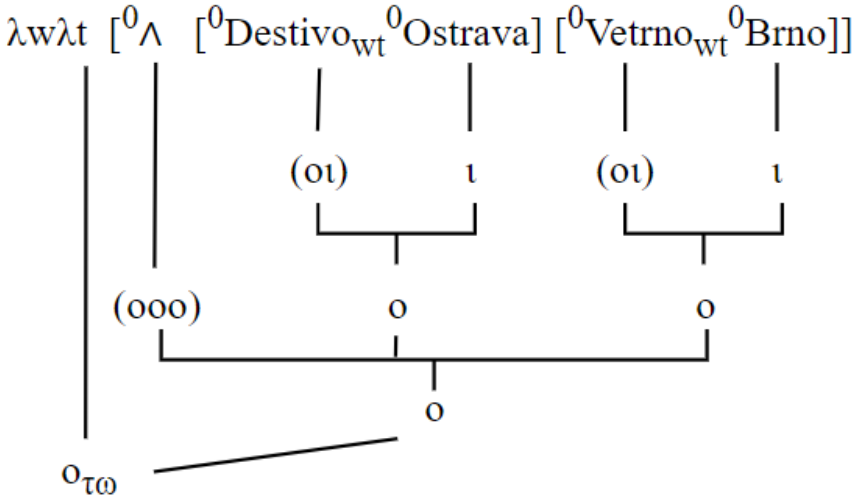
7.4. Parsování typů

Stejně jako u reprezentace typů a konstrukcí, k parsování typů přistoupíme podobně jako u parsování konstrukcí. Parsování typů je oproti konstrukcím poměrně jednoduché. Pokud řetězec typu obsahuje závorky, víme, že se jedná o molekulární typ, pokud ne, jde pouze o typ atomický, a pouze zkontrolujeme, jestli je tento typ v seznamu nám známých typů. Pokud se jedná o typ molekulární budeme tvořit strom typu. Pokud v průběhu zpracování molekulárního typu narazíme na podtyp, který je také molekulární musíme pro něj vytvořit nový strom, jehož kořen bude dítětem molekulárního typu, se kterým pracujeme.

Jako u konstrukcí i typy mohou obsahovat označení, že jejich součástí je i čas a svět. Proto před další prací s nimi musíme značky odstranit a strom typu upravit tak, aby tyto podtypy obsahoval. Například (Bool Indiv Indiv@tw)@tw převedeme na (((Bool Indiv ((Indiv Time) World)) Time) World).

Typ je následně přiřazený proměnným a objektům, kterých se týká podle definice kódu.

7.5. Typová kontrola



Obr. 7 - Vizualizace typové kontroly

Po parsování konstrukcí a typů můžeme proměnným a objektům přiřadit jejich typy. Následně začneme zjišťovat typy všech částí konstrukce, u kterých známe typy jejich podkonstrukcí. Takhle vlastně jdeme od listů stromu konstrukce až po kořen a po cestě určujeme typy specifické pro jednotlivé typy konstrukcí.

- Proměnné – proměnné mají přiřazený typ buď v kódu nebo v uzávěru, který se jich týká.
- Objekty – objekty jsou názvy, které jsou konstrukcí zmiňovány a musí na ně být volána trivializace
- Kompozice – typ kompozice je určen typy podkonstrukcí, které kompozici tvoří. Typicky je první podkonstrukcí konstrukce s molekulárním typem $(\beta\alpha_1 \ldots \alpha_n)$, a další podkonstrukce jsou typu $\alpha_1 \ldots \alpha_n$. Tedy jsou na první podkonstrukci, což je funkce, aplikovány argumenty a výstupním typem této funkce je typ β. Výstupní typ β je typem celé kompozice.
- Uzávěr – typ uzávěru je složen z typu kompozice, na který je uzávěr aplikován a typy jeho proměnných. Pokud má kompozice typ β a proměnné uzávěru mají typy $\alpha_1 \ldots \alpha_n$, poté typ celého uzávěru bude $(\beta\alpha_1 \ldots \alpha_n)$.
- Trivializace – trivializace objektu má stejný typ jako objekt. Trivializaci proměnné nebo kompozice přiřadíme typ vyššího řádu *.
- Dvojí provedení – dvojí provedení provádíme nad konstrukcí, která má typ *. Pokud známe typ, ze kterého byl typ vyššího řádu vytvořen, bude mít dvojí provedení výstupní typ původního typu. Tedy pokud na konstrukci s typem $(\beta\alpha_1 \ldots \alpha_n)$ aplikujeme trivializaci a dostaneme typ * a na tuto konstrukci aplikujeme dvojí provedení, výsledný typ dvojího provedení bude β.

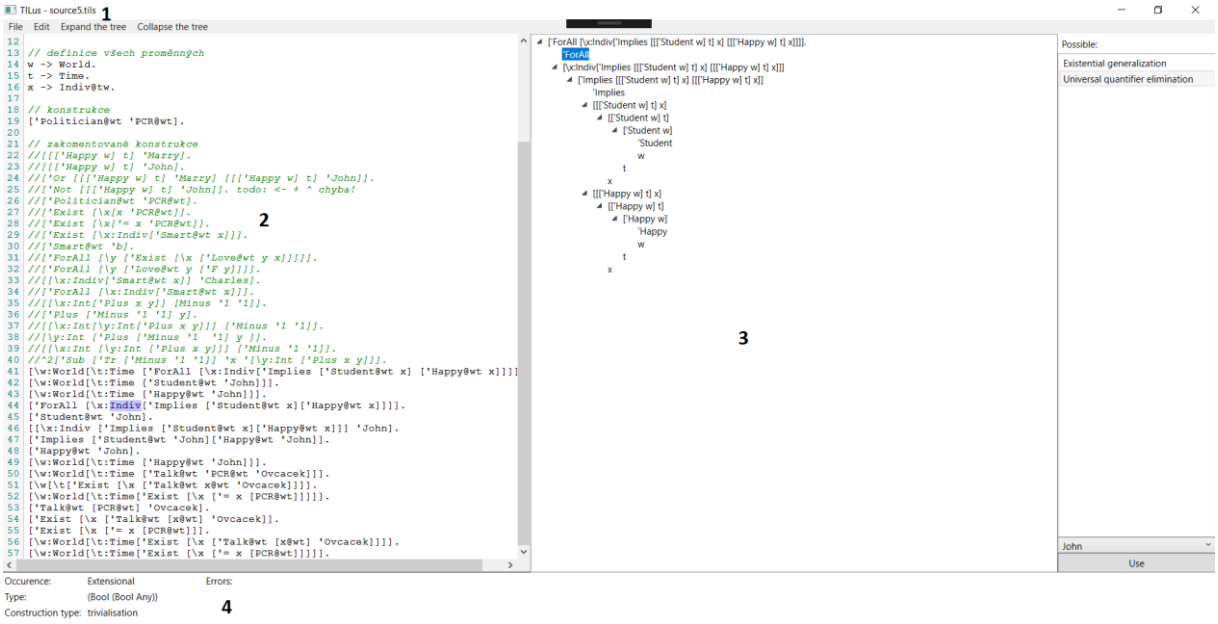
V průběhu aplikování typů kontrolujeme, zda sedí všechny typy funkcí a typy argumentů aplikovaných na funkci. Některé funkce mohou mít v rámci svého molekulárního typu i typ Any, který akceptuje jakýkoli typ. V TIL-Scriptu oproti TILu nerozeznáváme více druhů neurčitých typů.

7.6. Analýza kontextů

Posledním krokem v průběhu zpracování konstrukce je analýza kontextů. Již vytvořený strom konstrukce s typy procházíme od kořene k listům a u každé podkonstrukce určíme její kontext.

- Hyperintenzionální – hyperintenzionální kontext je nejjednodušší pro určení, protože ho můžeme jednoduše přiřadit všem trivializovaným kompozicím a všem dalším podkonstrukcím dané kompozice.
- Intenzionální – intenzionální kontext mají konstrukce, u kterých pracujeme přímo s nimi, a ne s jejich hodnotou. Pro náš algoritmus stačí, že to jsou všechny konstrukce, které nemají ani extenzionální, ani hyperintenzionální kontext.
- Extenzionální – u tohoto kontextu záleží na jeho pozici v kompozici, zda je nebo není v dosahu intenzionálního kontextu a jestli jsou správně aplikovány argumenty na funkci, která má tento kontext mít. Protože u konstrukce v extenzionálním kontextu pracujeme s její hodnotou, a ne s konstrukcí samotnou, musí být tato konstrukce funkce, na kterou jsou aplikovány všechny její argumenty. V dosahu intenzionálního kontextu se konstrukce nachází, pokud je její hodnota použita jako výstup uzávěru. Pokud tedy máme konstrukci $\ldots[\backslash x\ [A\ B_1\ B_2]]\ldots$, je konstrukce A v intenzionálním kontextu.

8. Uživatelské rozhraní



Obr. 8 - Screenshot programu s označenými částmi uživatelského rozhraní

Uživatelské rozhraní je rozděleno na pět částí. Tyto části jsou menu, editor, zobrazení stromu konstrukce, vlastnosti vybrané konstrukce a výběr možných operací.

- Menu – typické menu, pro načítání a ukládání kódů a pomocné metody pro editaci a otevření nebo zavření stromu konstrukce
- Editor – zde se píše veškerý kód, typy pro neklíčová slova, proměnné, i samotné konstrukce. Do kódu je možné zapisovat i komentáře za dvě lomítka. Editor zvládá nejenom ASCII, ale i Unicode znaky, a proto je možné psát komentáře i kód v češtině. Pokud je kurzor v editoru na některé konstrukci zobrazí se strom této konstrukce v pravém okně (pole 3). Pokud potřebujeme vybrat více konstrukcí, například pro operaci zavedení konjunkce je potřeba kliknout na číslo řádku dané konstrukce. Tím označíme danou konstrukci, a když jsou označeny dvě konstrukce zobrazí se v pravém okně a je možno na ně použít potřebné operace. Po každé změně kódu v editoru se kód zpracuje, aby bylo možno správně pracovat s konstrukcemi, případně napovídat u některých operací. Na pozici definici typů nezáleží, a proto záleží na uživateli, jestli napíše všechny typy na začátek kódu nebo až u konstrukce, kde se typ používá.
- Zobrazení stromu konstrukce – v tomto okně se zobrazuje strom právě vybrané konstrukce nebo konstrukcí. Je možné si zobrazit celý strom, tedy všechny podkonstrukce dané konstrukce. Při výběru konstrukce se v poli se seznamem operací (pole 5) zobrazí možné operace, které se dají aplikovat na konkrétní konstrukci. Vždy se zobrazí pouze operace, které jsou na dané konstrukci validní. Pro vybranou konstrukci se také zobrazí všechny její vlastnosti (pole 4).
- Zobrazení vlastností konstrukce – část okna ve spodní části ukazuje detailní informace o vybrané konstrukci. Konkrétně její kontext, celý typ, typ konstrukce a případné chyby nalezené při zpracování konstrukce.
- Seznam operací – v části okna nejvíce vpravo se zobrazuje seznam operací, které je možné vykonat na zrovna vybranou konstrukci, podkonstrukci nebo více konstrukcí. Všechny zobrazené operace jsou validní pro daný výběr. Pro některé operace se po vybrání zobrazí

seznam možností, který slouží jako další vstup dané operace. Například při eliminaci všeobecného kvantifikátoru je nutno vybrat, jakým prvkem bude nahrazena proměnná, které se kvantifikátor týká. Vstup je také kontrolován a zobrazí se pouze takové prvky, které odpovídají typu proměnné. Vybraná operace se provede po kliknutí na tlačítko "Use" a nově vytvořená konstrukce se poté vypíše na nový řádek v editoru.

9. Příklady použití

9.1. Příklad důkazu úsudku

Všichni psi štěkají nebo koušou.

Alík je pes a nekouše.

Někdo se bojí psů, kteří štěkají.

Někdo se bojí Alíka.

Důkaz v TIL:

1) $\forall x\ [[{}^0\text{Pes}_{wt}\ x] \supset [[{}^0\text{Štěká}_{wt}\ x] \vee [{}^0\text{Kouše}_{wt}\ x]]]$ 1. předpoklad
2) $[{}^0\text{Pes}_{wt}\ {}^0\text{Alík}] \wedge \neg[{}^0\text{Kouše}_{wt}\ {}^0\text{Alík}]$ 2. předpoklad
3) $\forall y\ [[[{}^0\text{Pes}_{wt}\ y] \wedge [{}^0\text{Štěká}_{wt}\ y]] \supset \exists x\ [{}^0\text{Bojí\_se}_{wt}\ x\ y]]$ 3. předpoklad
4) $[{}^0\text{Pes}_{wt}\ {}^0\text{Alík}]$ eliminace konjunkce, 2)
5) $\neg[{}^0\text{Kouše}_{wt}\ {}^0\text{Alík}]$ eliminace konjunkce, 2)
6) $[{}^0\text{Pes}_{wt}\ {}^0\text{Alík}] \supset [[{}^0\text{Štěká}_{wt}\ {}^0\text{Alík}] \vee [{}^0\text{Kouše}_{wt}\ {}^0\text{Alík}]]$ eliminace všeobecného kvantifikátoru, 1), substituce x/Alík
7) $[[{}^0\text{Štěká}_{wt}\ {}^0\text{Alík}] \vee [{}^0\text{Kouše}_{wt}\ {}^0\text{Alík}]]$ modus ponens, 4), 6)
8) $[{}^0\text{Štěká}_{wt}\ {}^0\text{Alík}]$ eliminace disjunkce, 5), 7)
9) $[{}^0\text{Pes}_{wt}\ {}^0\text{Alík}] \wedge [{}^0\text{Štěká}_{wt}\ {}^0\text{Alík}]$ zavedení konjunkce, 4), 8)
10) $[[{}^0\text{Pes}_{wt}\ {}^0\text{Alík}] \wedge [{}^0\text{Štěká}_{wt}\ {}^0\text{Alík}]] \supset \exists x\ [{}^0\text{Bojí\_se}_{wt}\ x\ {}^0\text{Alík}]$ eliminace všeobecného kvantifikátoru, 3), substituce y/Alík
11) $\exists x\ [{}^0\text{Bojí\_se}_{wt}\ x\ {}^0\text{Alík}]$ modus ponens, 9), 10)

Typy: x, y → ι, Pes,Štěká,Kouše/$(o\iota)_{\tau\omega}$, Alík/ι

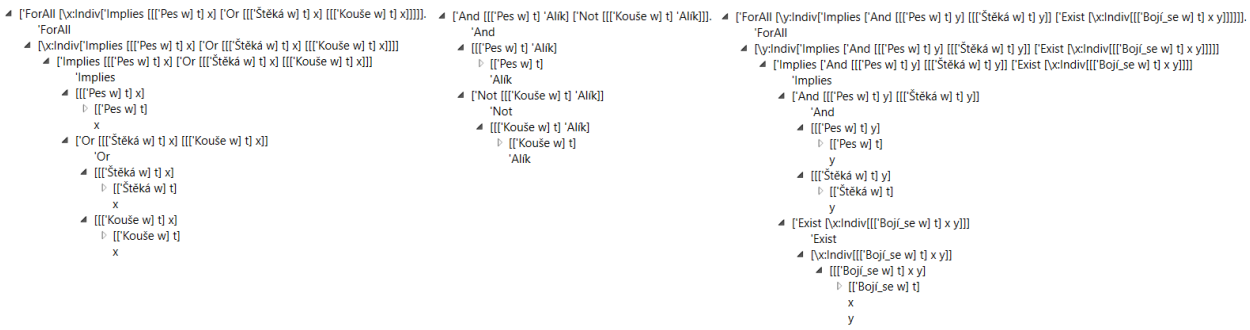
Důkaz v TIL-Scriptu a programu:

V programu definujeme typy takto:

```
Pes,Štěká,Kouše/(Bool Indiv)@tw.
Bojí_se/(Bool Indiv Indiv)@tw.
Alík/Indiv.
w->World.
t->Time.
x,y->Indiv.
```

Než začneme s důkazem musíme do programu zapsat typy objektů a proměnných a všechny předpoklady.

1) `['ForAll [\x ['Implies ['Pes@wt x] ['Or ['Štěká@wt x] ['Kouše@wt x]]]]].` 1. předpoklad
2) `['And ['Pes@wt 'Alík] ['Not ['Kouše@wt 'Alík]]].` 2. předpoklad
3) `['ForAll [\y ['Implies ['And ['Pes@wt y] ['Štěká@wt y]] ['Exist [\x ['Bojí_se@wt x y]]]]]].` 3.předpoklad



Obr. 9 - 1. důkaz, stromy konstrukce předpokladů v programu

Na druhý předpoklad můžeme použít pravidlo eliminace konjunkce. Vybereme řádek s předpokladem a zobrazí se strom konstrukce. Poté co vybereme konstrukci s konjunkcí zobrazí se možnost eliminace konjunkce (Conjunction elimination). Tlačítkem use se pravidlo použije.

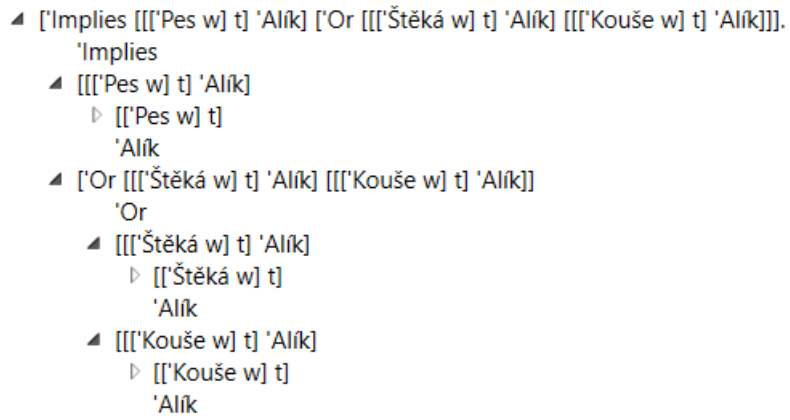
4) ['Pes@wt 'Alík]. eliminace konjunkce, 2)
5) ['Not ['Kouše@wt 'Alík]]. eliminace konjunkce, 2)

Jako další krok v důkazu potřebujeme eliminovat všeobecný kvantifikátor v prvním předpokladu. Zvolíme předpoklad, následně zvolíme kvantifikátor, který chceme eliminovat – v tomto případě pouze jediný 'ForAll. Poté vybereme pravidlo eliminace všeobecného kvantifikátoru (Universal quantifier elimination) a nad tlačítkem „Use“ vybereme prvek, kterým chceme nahradit proměnnou. Aby se prvek zobrazil musí mít stejný typ jako je typ uzávěru, nad kterým je kvantifikátor.

6) [[\x:Indiv['Implies [[['Pes w] t] x] ['Or [[['Štěká w] t] x] [[['Kouše w] t] x]]]] 'Alík]. eliminace všeobecného kvantifikátoru, 1)

Oproti pravidlům v TIL-u v TIL-Scriptu nedojde k přímému nahrazení proměnné všeobecného kvantifikátoru za zvolený prvek, ale pouze se prvek aplikuje jako atribut funkce tvořený uzávěrem s proměnnou. Následně musíme provést beta redukci jménem (Beta by name), pokud to jde. Vybereme uzávěr ve stromu konstrukce a použijeme dané pravidlo.

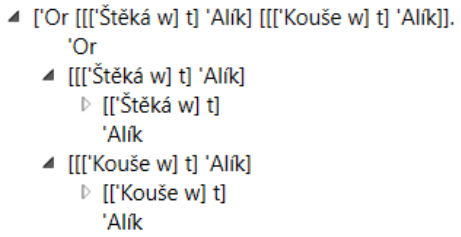
7) ['Implies ['Pes@wt 'Alík] ['Or ['Štěká@wt 'Alík] ['Kouše@wt 'Alík]]]. beta redukce jménem, 6)



Obr. 10 - strom konstrukce výrazu "Pokud je Alík pes, pak štěká nebo kouše"

Když už víme, že Alík je pes a to, že je pes implikuje, že štěká nebo kouše, můžeme použít pravidlo eliminace implikace (Implication elimination). Pro toto pravidlo musíme vybrat dvě formule, a to můžeme udělat pomocí levé lišty v editoru, kde klikneme na číslo daných formulí. Poté co budeme mít vybrané obě formule, zobrazí se nám všechna pravidla, která na ně oboje můžeme použít.

8) ['Or ['Štěká@wt 'Alík] ['Kouše@wt 'Alík]]. modus ponens, 4), 6)

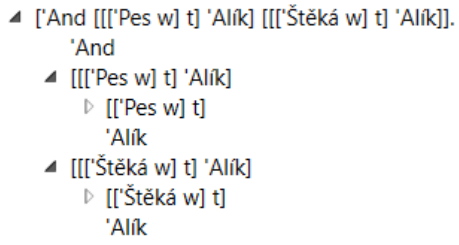


Obr. 11 - Strom konstrukce výrazu "Alík je štěká nebo kouše"

Protože víme, že Alík nekouše, můžeme pomocí eliminace disjunkce (Disjunction elimination) vydedukovat, že Alík štěká. Opět musíme pro zobrazení pravidla vybrat obě formule.

9) ['Štěká@wt 'Alík]. eliminace disjunkce, 5), 7)
Pro zavedení konjunkce (Conjunction introduction) vybereme dvě formule, na které chceme konjunkci aplikovat. Podle pořadí výběru formulí bude vybráno pořadí formulí ve výsledné konjunkci.

10) ['And ['Pes@wt 'Alík] ['Štěká@wt 'Alík]]. zavedení konjunkce, 4), 8)



Obr. 12 - Strom konstrukce výrazu "Alík je pes a štěká"

11) [[\y ['Implies ['And ['Pes@wt y] ['Štěká@wt y]] ['Exist [\x ['Bojí_se@wt x y]]] 'Alík]. eliminace všeobecného kvantifikátoru, 3)

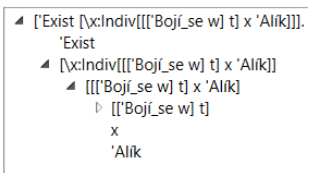
Nyní eliminujeme všeobecný kvantifikátor z třetího předpokladu, ať můžeme odvodit, jestli se někdo bojí Alíka.

12) ['Implies ['And ['Pes@wt 'Alík] ['Štěká@wt 'Alík]] ['Exist [\x ['Bojí_se@wt x 'Alík]]]]. eliminace všeobecného kvantifikátoru, 3), substituce y/Alík

Jak již bylo výše uvedeno, po eliminaci všeobecného kvantifikátoru musíme udělat beta redukci jménem, pokud je to možné.

13) ['Exist [\x ['Bojí_se@wt x 'Alík]]]. modus ponens, 9), 10)

```
1  //Všichni psi štěkají nebo koušou.
2  //Alík je pes a nekouše.
3  //Někdo se bojí psů, kteří štěkají.
4  //------------
5  //Někdo se bojí Alíka.
6
7  Pes,Štěká,Kouše/(Bool Indiv)@tw.
8  Bojí_se/(Bool Indiv Indiv)@tw.
9  Alík/Indiv.
10 w->World.
11 t->Time.
12 x,y->Indiv.
13
14 ['ForAll [\x:Indiv ['Implies ['Pes@wt x] ['Or ['Štěká@wt x] ['Kouše@wt x]]]].
15 ['And ['Pes@wt 'Alík] ['Not ['Kouše@wt 'Alík]]].
16 ['ForAll [\y ['Implies ['And ['Pes@wt y] ['Štěká@wt y]] ['Exist [\x ['Bojí_se@wt
17 [[['Pes w] t] 'Alík].
18 ['Not [[['Kouše w] t] 'Alík]].
19 [[\x:Indiv['Implies [[['Pes w] t] x] ['Or [[['Štěká w] t] x] [[['Kouše w] t] x]]
20 ['Implies [[['Pes w] t] 'Alík] ['Or [[['Štěká w] t] 'Alík] [[['Kouše w] t] 'Alík
21 ['Or [[['Štěká w] t] 'Alík] [[['Kouše w] t] 'Alík]].
22 [[['Štěká w] t] 'Alík].
23 ['And [[['Pes w] t] 'Alík] [[['Štěká w] t] 'Alík]].
24 [[\y:Indiv['Implies ['And [[['Pes w] t] y] [[['Štěká w] t] y]] ['Exist [\x:Indiv
25 ['Implies ['And [[['Pes w] t] 'Alík] [[['Štěká w] t] 'Alík]] ['Exist [\x:Indiv[[
```



Obr. 13 - Postup prvního důkazu

9.2. Příklad důkazu tautologie

Pokud všichni, kdo jsou chytří a ambiciózní jsou bohatí a všichni kdo nejsou hloupí jsou chytří a existuje někdo, kdo je ambiciózní a není hloupý, pak existuje někdo, kdo je bohatý.

Důkaz v TIL:

1) $[\forall x[[[^0\text{Chytrý}_{wt}\ x] \wedge [^0\text{Ambiciózní}_{wt}\ x]] \supset [^0\text{Bohatý}_{wt}\ x]] \wedge \forall x[\neg[^0\text{Hloupý}_{wt}\ x] \supset [^0\text{Chytrý}_{wt}\ x]] \wedge \exists x[[^0\text{Ambiciózní}_{wt}\ x] \wedge \neg[^0\text{Hloupý}_{wt}\ x]]]$ předpoklad
2) $\forall x[[[^0\text{Chytrý}_{wt}\ x] \wedge [^0\text{Ambiciózní}_{wt}\ x]] \supset [^0\text{Bohatý}_{wt}\ x]]$ eliminace konjunkce, 1)
3) $\forall x[\neg[^0\text{Hloupý}_{wt}\ x] \supset [^0\text{Chytrý}_{wt}\ x]]$ eliminace konjunkce, 1)
4) $\exists x[[^0\text{Ambiciózní}_{wt}\ x] \wedge \neg[^0\text{Hloupý}_{wt}\ x]]$ eliminace konjunkce, 1)
5) $[^0\text{Ambiciózní}_{wt}\ {}^0a] \wedge \neg[^0\text{Hloupý}_{wt}\ {}^0a]$ eliminace existenčního kvantifikátoru, 4)
6) $[^0\text{Ambiciózní}_{wt}\ {}^0a]$ eliminace konjunkce, 5)
7) $\neg[^0\text{Hloupý}_{wt}\ {}^0a]$ eliminace konjunkce, 5)
8) $\neg[^0\text{Hloupý}_{wt}\ {}^0a] \supset [^0\text{Chytrý}_{wt}\ {}^0a]$ eliminace všeobecného kvantifikátoru, 3), substituce x/a
9) $[^0\text{Chytrý}_{wt}\ {}^0a]$ eliminace disjunkce, 7), 8)
10) $[[^0\text{Chytrý}_{wt}\ {}^0a] \wedge [^0\text{Ambiciózní}_{wt}\ {}^0a]] \supset [^0\text{Bohatý}_{wt}\ {}^0a]$ eliminace všeobecného kvantifikátoru, 1), substituce x/a
11) $[^0\text{Chytrý}_{wt}\ {}^0a] \wedge [^0\text{Ambiciózní}_{wt}\ {}^0a]$ zavedení konjunkce, 6), 9)
12) $[^0\text{Bohatý}_{wt}\ 'a]$ modus ponens, 10), 11)
13) $\exists x[^0\text{Bohatý}_{wt}\ x]$ zavedení existenčního kvantifikátoru, 12)

Typy: $x \rightarrow \iota$, Chytrý,Ambiciózní,Bohatý,Hloupý/$(o\iota)_{\tau\omega}$, a/ι

Důkaz v TIL-Scriptu a programu:

První definujeme typy:
Chytrý,Ambiciózní,Bohatý,Hloupý /(Bool Indiv)@tw.
w->World.
t->Time.
x->Indiv.

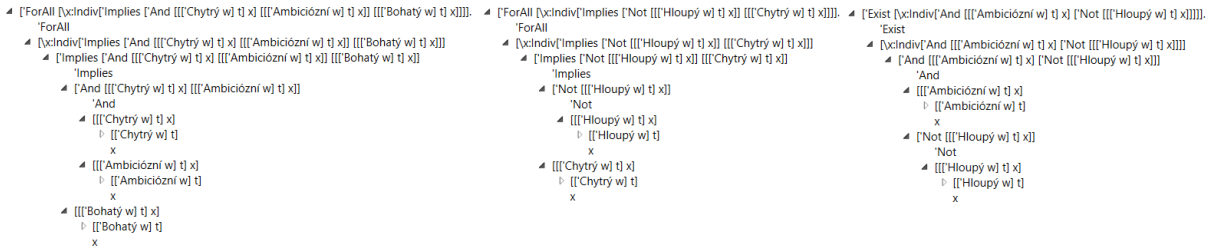
1) ['And ['And ['ForAll [\x ['Implies ['And ['Chytrý@wt x] ['Ambiciózní@wt x]] ['Bohatý@wt x]]]] ['ForAll [\x ['Implies ['Not ['Hloupý@wt x]] ['Chytrý@wt x]]]]] ['Exist [\x ['And ['Ambiciózní@wt x] ['Not ['Hloupý@wt x]]]]]]. předpoklad
   Protože v TIL-Scriptu nelze vytvořit konjunkce s více než dvěmi členy musíme předpoklad s třemi formulemi spojenými konjunkcí spojit pomocí dvou do sebe vnořených konjunkcí. Následně musíme pomocí eliminace konjunkce eliminovat vnější konjunkci a následně konjunkci vnitřní.

2) ['And ['ForAll [\x ['Implies ['And ['Chytrý@wt x] ['Ambiciózní@wt x]] ['Bohatý@wt x]]]] ['ForAll [\x ['Implies ['Not ['Hloupý@wt x]] ['Chytrý@wt x]]]]].
   eliminace konjunkce, 1)

   Vnitřní konjunkce, kterou eliminujeme dále.
3) ['Exist [\x ['And ['Ambiciózní@wt x] ['Not ['Hloupý@wt x]]]]].
   eliminace konjunkce, 1)

4) ['ForAll [\x ['Implies ['And ['Chytrý@wt x] ['Ambiciózní@wt x]] ['Bohatý@wt x]]]]. eliminace konjunkce, 2)

5) ['ForAll [\x ['Implies ['Not ['Hloupý@wt x]] ['Chytrý@wt x]]]]. eliminace konjunkce, 2)



Obr. 14 - Stromy konstrukcí předpokladů

Po extrakci všech členů předpokladu můžeme začít eliminovat kvantifikátory. Jako první eliminujeme existenční kvantifikátor (Existential quantifier elimination), který vytvoří nový prvek, který ještě neexistuje. Proto, aby byl důkaz platný, musíme do konce důkazu opět kvantifikátor zavést. Po použití pravidla musíme definovat typ nového prvku, pokud bude nový prvek 'a jako v příkladu dopíšeme „a/Indiv".

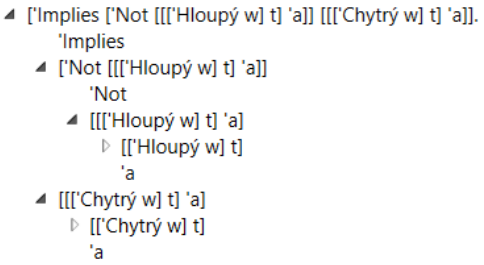
6) ['And [[['Ambiciózní w] t] 'a] ['Not [[['Hloupý w] t] 'a]]]. eliminace existenčního kvantifikátoru, 3)

7) [[['Ambiciózní w] t] 'a]. eliminace konjunkce, 6)

8) ['Not [[['Hloupý w] t] 'a]]. eliminace konjunkce, 6)

9) [[\x:Indiv['Implies ['Not [[['Hloupý w] t] x]] [[['Chytrý w] t] x]]] 'a]. eliminace všeobecného kvantifikátoru, 5)

10) ['Implies ['Not [[['Hloupý w] t] 'a]] [[['Chytrý w] t] 'a]]. beta redukce jménem, 9)



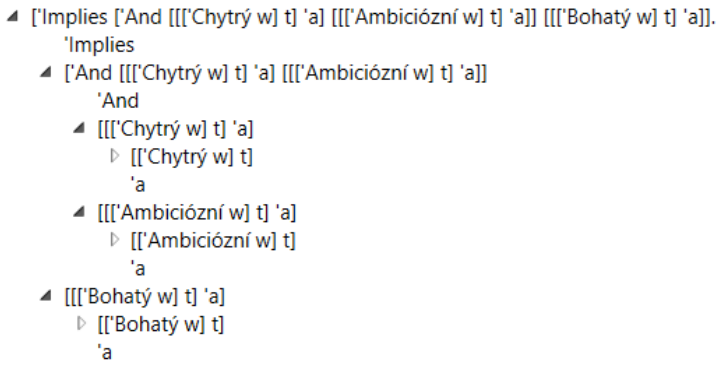
Obr. 15 - Strom konstrukce výrazu "Pokud prvek a není hloupý, pak je chytrý"

11) [[['Chytrý w] t] 'a]. modus ponens, 8), 10)

12) ['And [[['Chytrý w] t] 'a] [[['Ambiciózní w] t] 'a]]. zavedení konjunkce, 7), 11)

13) [[\x:Indiv['Implies ['And [[['Chytrý w] t] x]] [[['Ambiciózní w] t] x]] [[['Bohatý w] t] x]]] 'a]. eliminace všeobecného kvantifikátoru, 4)

14) ['Implies ['And [[['Chytrý w] t] 'a] [[['Ambiciózní w] t] 'a]] [[['Bohatý w] t] 'a]].
beta redukce jménem, 13)

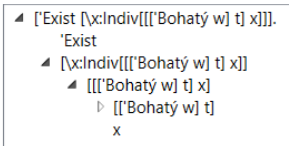


Obr. 16 - Strom konstrukce výrazu "Pokud je prvek a chytrý a ambiciózní, pak je bohatý"

15) [[['Bohatý w] t] 'a]. modus ponens, 12), 14)
Pro zavedení existenčního kvantifikátoru (Existential generalization) musíme vybrat danou podkonstrukci, na kterou konkrétně chceme kvantifikátor aplikovat. V příkladu tedy vybereme řádek v posledním kroku důkazu a zvolíme podkonstrukci 'a, na kterou aplikujeme pravidlo.

16) ['Exist [\x:Indiv[[['Bohatý w] t] x]]]. zavedení existenčního kvantifikátoru, 15)

```
1  //Pokud všichni, kdo jsou chytří a ambiciózní jsou bohatí a
2  //  všichni kdo nejsou hloupí jsou chytří a existuje někdo,
3  //  kdo je ambiciózní a není hloupý, pak existuje někdo, kdo je bohatý.
4
5  Chytrý,Ambiciózní,Hloupý,Bohatý/(Bool Indiv)@tw.
6  a/Indiv.
7  x->Indiv.
8  w->World.
9  t->Time.
10
11 ['And ['And ['ForAll [\x:Indiv['Implies ['And [[['Chytrý w] t] x] [[['Ambiciózní
12 ['And ['ForAll [\x:Indiv['Implies ['And [[['Chytrý w] t] x] [[['Ambiciózní w] t]
13 ['Exist [\x ['And ['Ambiciózní@wt x] ['Not ['Hloupý@wt x]]]]].
14 ['ForAll [\x ['Implies ['And ['Chytrý@wt x] ['Ambiciózní@wt x]] ['Bohatý@wt x]]]
15 ['ForAll [\x ['Implies ['Not ['Hloupý@wt x]] ['Chytrý@wt x]]]].
16 ['And [[['Ambiciózní w] t] 'a] ['Not [[['Hloupý w] t] 'a]]].
17 [[['Ambiciózní w] t] 'a].
18 ['Not [[['Hloupý w] t] 'a]].
19 [[\x:Indiv['Implies ['Not [[['Hloupý w] t] x]] [[['Chytrý w] t] x]]] 'a].
20 ['Implies ['Not [[['Hloupý w] t] 'a]] [[['Chytrý w] t] 'a]].
21 [[['Chytrý w] t] 'a].
22 ['And [[['Chytrý w] t] 'a] [[['Ambiciózní w] t] 'a]].
23 [[\x:Indiv['Implies ['And [[['Chytrý w] t] x] [[['Ambiciózní w] t] x]] [[['Bohat
24 ['Implies ['And [[['Chytrý w] t] 'a] [[['Ambiciózní w] t] 'a]] [[['Bohatý w] t]
25 [[['Bohatý w] t] 'a].
26 ['Exist [\x:Indiv[[['Bohatý w] t] x]]].
```



Obr. 17 - Celý druhý důkaz

10. Závěr

Výsledný program je možné použít pro zrychlení a zjednodušení provádění důkazu v jazyce TIL. Díky automatickému zobrazování pouze pravidel, které je možno v danou chvíli na konkrétní konstrukce použít, omezuje výskyt chyb špatnou aplikací pravidla, které není validní. Dále automatickým vyhodnocením pravidla eliminuje chyby způsobené špatným přepisem nebo nepozorností člověka. Pomocí automatické kontroly typů si může uživatel snadno ověřit, jestli mají konstrukce validní typy, a jestli sedí kontext konstrukcí.

Přesto, že je program funkční, je stále nutné ručně vybrat konstrukce a pravidla, která chceme použít. Dalším krokem vývoje programu by tedy bylo automatické navrhování relevantních pravidel, které posunou důkaz dále. Tímto způsobem by se dalo program vylepšovat, aby byl co nejvíce automatický a na základě předpokladů a závěru by mohl závěr dokázat sám, případně by mohl generovat vše, co z předpokladů vyplývá.

Literatura

1. DUŽÍ, Marie a Pavel MATERNA. TIL jako procedurální logika: Průvodce zvídavého čtenáře Transparentní intensionální logikou. Bratislava: Aleph, 2012. ISBN 978-80-89491-08-7.
2. CIPRICH, Nikola, Marie DUŽÍ a Michal KOŠINÁR. The TIL-Script Language [online]. [cit. 2018-05-31]. Dostupné z: https://www.academia.edu/17035450/The_TIL-Script_Language
3. DUŽÍ, Marie. Logika pro informatiky (a příbuzné obory): učební text. Ostrava: VŠB-TU Ostrava, 2012. ISBN 978-80-248-2662-2.
4. DUŽÍ, M., MENŠÍK, M., PAJR, M., PATSCHKA, V. Natural deduction system in the TIL-script language. In Frontiers in Artificial Intelligence and Applications. IOS Press, 2019, s. 237-255.
5. DUŽÍ, Marie, Bjørn JESPERSEN a Pavel MATERNA. Procedural semantics for hyperintensional logic: foundations and applications of transparent intensional logic. New York: Springer, c2010. ISBN 978-90-481-8811-6.
6. DUŽÍ, M., MENŠÍK, M. Inferring knowledge from textual data by natural deduction. *Computación y Sistemas*, 2019, (to appear)
7. MENŠÍK, M., KERMASCHEK, J., CIENCIALA, L. Existential Generalization in TIL. In *17th International Multidisciplinary Scientific GeoConference: SGEM 2017 : conference proceedings : 29 June-5 July, 2017, Albena, Bulgaria. Volume 17. Issue 21*. Sofia : STEF92 Technology Ltd., 2017, s. 311-318.
8. DUŽÍ, M., FAIT, M., MENŠÍK, M. Context recognition for a hyperintensional inference machine. In *AIP Conference Proceedings. Volume 1863*. Melville : American Institute of Physics, 2017, s. nestránkováno.
9. DUŽÍ, M., MENŠÍK, M., ČÍHALOVÁ, M., PERDEK, M. Hyperintensional, Intensional and Extensional Context Recognition. In *Mendel 2013 : 19th International Conference on Soft Computing : June 26-28, 2013, Brno, Czech Republic*. Brno : Brno University of Technology, 2013, s. 427-432.

Seznam příloh

Příloha v IS Edison

Obsah:

- Zdrojové kódy
- Program